Theoretische Informatik I

Georg Fischer Informatik Diplom, Semester 1 28.06.03

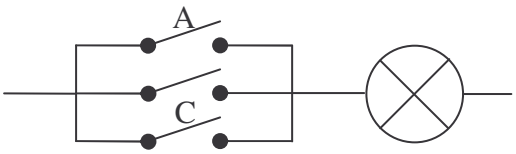
Was ist ein Computer?

- Informationen nach Vorgaben verarbeiten
- Reine Berechnungen
- Ein-/Ausgabe (In-/Output)
- Daten speichern
- Flexibel, nicht nur eine Anwendung
- Steuert Peripherie

= Der Computer ist programmierbar und ein Rechner

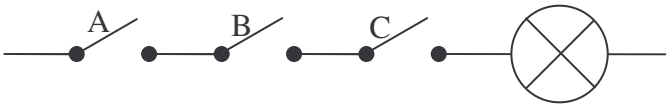
Definition Computer:
Ein Computer ist ein technisches „Universalgerät“, das alle Formen unseres kausal-logischen Denkens abbilden und auf elektronische/mechanische Weise nachvollziehen kann.
Beispiel:
Verein #1: Vertrauen:

- Unterschriftsbevollmächtigte Mitglieder: A, B, C
- Einer der drei Bevollmächtigten genügt, also A oder B oder C
  Schaltungsbeispiel:



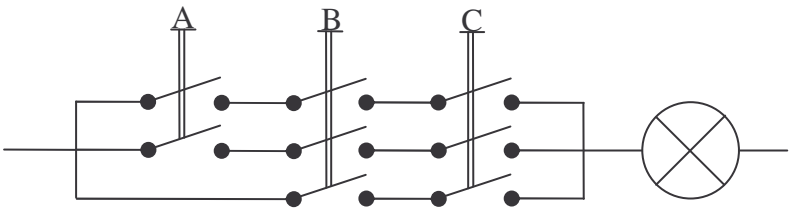
Verein #2: Misstrauen

- Unterschriftsbevollmächtigte Mitglieder: A, B, C
- Alle drei müssen unterschreiben, also A und B und C
  Schaltungsbeispiel:



Verein #3: „Goldene Mitte“

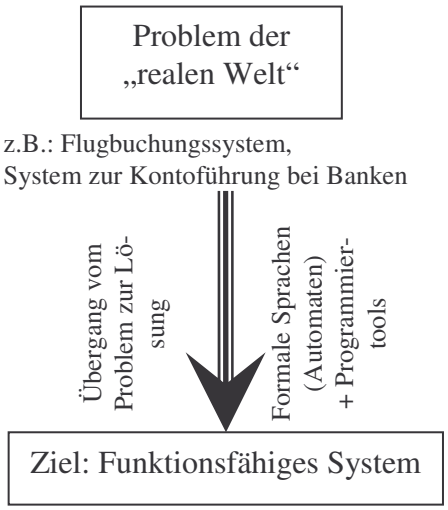
- Unterschriftsbevollmächtigte Mitglieder: A, B, C
- Zwei Unterschriften sind nötig.
  Schaltungsbeispiel:



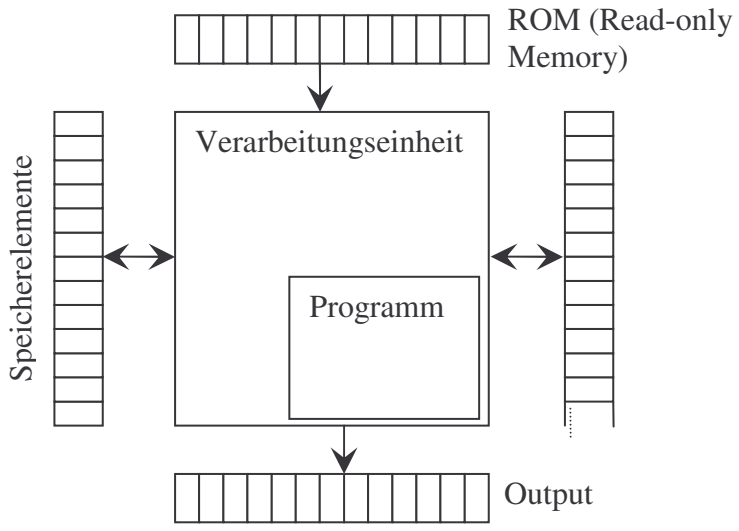
Beispiel einer Wahrheitstafel zum Verein „Misstrauen“ (0 = falsch/ungültig; 1 = richtig/gültig)

Mögliche Konstellationen			Unterschriften gültig/rechtskräftig?
A	B	C	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Was ist die Informatik?



Allgemeines Modell eines Automaten



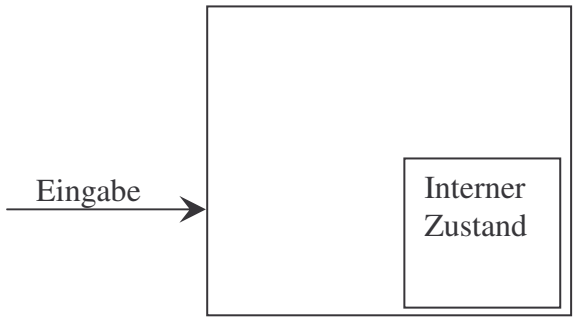
Wichtige Themen beim Automaten:

- Speicherkapazität
- Zugriffsarten (z.B.: Schreiben oder lesen; wahlfreier Zugriff oder bestimmte Reihenfolge;)

- Deterministische und nicht-deterministische Automaten
  Deterministische Automaten arbeiten sämtliche Vorgänge linear ab
  Beispiel: Ein Stapel Bierkästen. Man nimmt erst die Flaschen aus dem oberstem Kasten. Ist dieser leer, nimmt man den Kasten weg und bedient sich aus den Flaschen des Kastens darunter… und so weiter…
  Nicht-Deterministische Automaten hingegen arbeiten sämtliche Vorgänge nicht-linear ab
  Beispiel: Eine Schüssel voller Lose. Man wühlt und greift sich eins… dann wühlt man wieder und nimmt ein zweites… und so weiter...
- Ende der Bedingungen

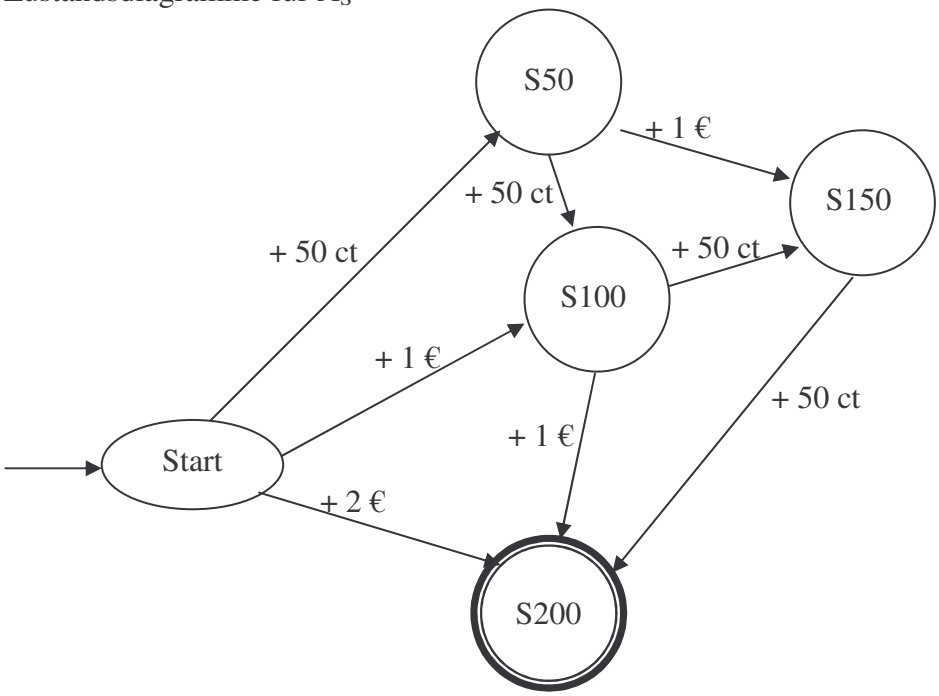
Endliche Automaten

- Jedes Automatenmodell hat eine endliche Gedächtnisleistung, d.h. nur endlich viele interne Zustände.



  In Abhängigkeit von der Eingabe und dem aktuellem Zustand geht der Automat in einen neuen Zustand über.
- Endzustände
  Beispiel: Ein Schwimmbadautomat à A_S
  Aufgaben:
  - Eintritt kontrollieren
  - Eintritt = 2 €
  - Akzeptable Geldstücke: 0,50 €; 1,00 €; 2,00 €;
  - Einfach, aber kundenfreundlich: „Wenn min. 2,00 € bezahlt sind, dann öffnet sich die Schranke

- Zustandsdiagramme für A_S



Zustand: ¡ (Ein Kreis)
Startzustand: à ¡ (Ein Pfeil, der auf einen Kreis mit der Inschrift „Start“ deutet)
Endzustand: © (Doppelter Kreis)
Zustandsübergang: à (Ein Pfeil)

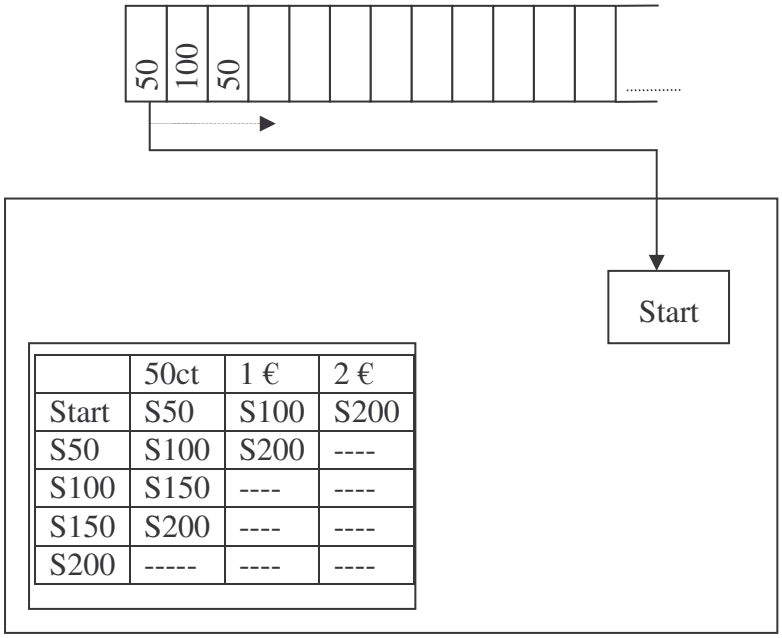
- Ein Speicher

Eingabespeicher:

Nach rechts unbeschränkt offen (angedeutet durch die Punkte rechts außen)
  - Zeichen werden von links nach rechts gelesen
  - Nur Lese-Zugriff (ROM)

Zustandsspeicher:
  - Eine Speicherzelle
  - Aktueller Zustand
  - Endlich viele Zustände

Black-Box-Modell



	50ct	1 €	2 €
Start	S50	S100	S200
S50	S100	S200	----
S100	S150	----	----
S150	S200	----	----
S200	-----	----	----

Die Eingabefolge wird vom Automatakzeptiert, falls sie komplett gelesen wird, d.h. der Lesekopf befindet sich auf der Speicherzelle hinter der Eingabefolge und der Zustandsspeicher beinhaltet einen Endzustand.

Definition:

Die Menge der vom Automaten akzeptierten Eingabefolgen heißt „Die Sprache des Automaten“

Beispiel:

Menge{50 100 50, 100 100, 100 200, 200, 50 50 50 50, 100 50 50, 50 100 200, 50 200, 50 50 200, 50 50 50 200, 50 50 100, 50 100 100, 50 100 100, 50 50 50 100, 100 50 200, 100 50 100}

Alphabete, Worte, Sprachen

Alphabete

Symbole, aus denen die Eingabewörter bestehen, fassen wir zu einer Menge, dem „Eingabealphabet“ zusammen.

Σ = Zeichen für Eingabealphabete

Beispiel

A: Σ = {50 100 50}

B: Σ = {0, 1}

Allgemein:

$\Sigma = \{a_1, a_2, \ldots, a_n\}$ ß n = ≥ 0

wenn n = 0, dann ist $\Sigma = \varnothing$ (leere Menge)

Definition:

Alphabetische Ordnung:

$a_i < a_{i+1}$ für $1 \leq i \leq n = 1$

Worte

Definition:

1. Jeder Buchstabe $a \in \Sigma$ ist auch Wert über dem Eingabealphabet Σ^* und somit ein Wort
2. v, $w \in \Sigma^*$, dann ist auch die Verkettung ein Wort (z.B.: vw) à vw $\in \Sigma^*$
3. ε (Epsilon), das leere Wort, das aus Null Buchstaben besteht, ist immer ein Wort über Σ^* à $\varepsilon \in \Sigma^*$

Bemerkung:

ε w = w ε = w à gilt für alle w ∈ Σ^*
Σ = Σ* - {ε} (Alphabete sind Worte, ohne „das leere Wort“)
Beispiel hierfür:
Σ = {a, b}
Σ* = {ε, a, b, aa, ab, bb, ba, ... }
$\Sigma = \{a_1, a_2, a_3, \ldots, a_n\}$
$\Sigma^* = \{\varepsilon, a_1, a_1a_2, a_1a_3, \ldots\}$
Bezeichnungen:
w = xyz (sprich: Wort ist gleich xyz)
w ∈ Σ* = x,y,z ∈ Σ*
x = der Präfix, y = der Infix, z = der Suffix oder Postfix
insbesondere können x oder z = ε sein !

Definition:
Lexikografische Ordnung auf Σ*:
$\prec \subseteq \Sigma^* \times \Sigma^* = \{v, w \mid v \in \Sigma^*, w \in \Sigma^*\}$ (Wortpaare; zwei oder mehr Wörter werden zusammengefasst

$v = v_1 \ldots v_m \quad v_1 \in \Sigma^*; v_i \in \Sigma^* \quad 1 \le i \le m$

$w = w_1 \ldots w_n \quad w_1 \in \Sigma^*; w_j \in \Sigma^* \quad 1 \le j \le n$

$v \prec w$ gdw (genau dann, wenn)
Beispiel:
Σ = {a,b} a < b (a vor b, alles sei gegeben)
1. $bb \prec aaa$ (mengenmäßig größer)
2. $baab \prec baba$ (nächsthöhere Stelle)
Beispiel anhand unseres Automaten:
{200, 50 200, 100 100, 100 200, 50 50 100, …}

Wortfunktionen

Erstes Beispiel: Suchfunktion eines Texteditors

$$substr(v,w) = \left\{ \frac{ja, falls\ v\ \ Infix\ von\ w}{nein \quad sonst} \right\}$$ (substr = SubString)

substr: Σ^+ x Σ^+ à {ja, nein} (à = ordne zu)
substr: (001010, 01) = ja
substr: (001010, 011) = nein

zweites Beispiel: Länge eines Wortes $W_0 = \{0,1,2,\ldots\}$

$l = \Sigma^* \to N_0$

$l_\varepsilon = 0$

$w \in \Sigma^*, a \in \Sigma$

$l(wa) = l(w) + 1$

l(gesamtes Wort)=l(Wort, reduziert um einen Buchstaben) + 1

Beispiel:

∑={a,b}, aba (ß das Wort, dessen Buchstaben, die in der { } definiert sind, gezählt werden sollen)

l(aba) = l(ab) + 1 = (l(a) + 1) + 1 = ((l(ε) + 1) + 1) + 1) = 3 (da l(ε) = 0)

drittes Beispiel:

anzahl ∑* x ∑ = N (ß natürliche Zahl)

anzahl (ε, b) = 0 (b ∈ ∑

$w \in \Sigma^* \; a \in \Sigma$

$$anzahl(wa,b) = \left\{ \frac{anzahl(wb)+1, falls\, a = b}{anzahl(wb)\; sonst} \right\}$$

∑{a,b}

anzahl(ababb, b) = anzahl (abab,b) + 1 = anzahl (aba, b) + 2 = anzahl (aa, b) + 3 = anzahl (ε, b) + 3 = 3

viertes Beispiel: Konkatenation

$\circ = \Sigma^* x \Sigma^* \to \Sigma^*$

$\circ(v,w) = vw = v \circ x (v\ komkidiert\ mit\ w)$

fünftes Beispiel: „Tausche“

$tausche\ \ \Sigma^* x \Sigma x \Sigma \to \Sigma^*$

$tausche(\varepsilon, a, b) = \varepsilon\ für\ alle\ \ a,b \in \Sigma$

$$tausche(cw,a,b) = \left\{ \frac{b \circ tausche(w,a,b)\, falls\, c = a}{c \circ tausche(w,a,b)\, falls\, c \neq a} \right\}$$

tausche (Wort, ersetze das, durch das)

Beispiel hierfür:

Σ = {a,2,3}

tausche(12132,2,3) = 1°tausche(2132,2,3) = 1°3°tausche(132,2,3) = 1°3°1°tausche(32,2,3)=

1°3°1°3°tausche(2,2,3) = 1°3°1°3°3°ε = 13133

Formale Sprachen

Definition: Σ = Alphabet

$L \subseteq \Sigma^*$ (=formale Sprache über Σ^*

Mengen von Wörtern:

∩ = Schnittmenge (z.B.: $L_1 \cap L_2$)

∪ = Vereinigungsmenge (z.B.: $L_1 \cup L_2$)

- = Differenz

Konkatenation von Sprachen

L_1, L_2 = formale Sprachen

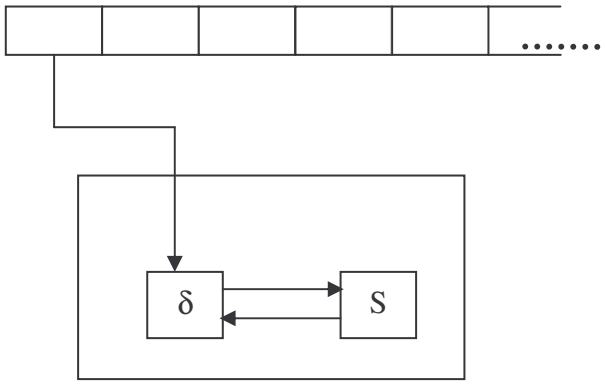
$L_1 \circ L_2 = \{v \circ w \mid v \in L_1, w \in L_2\}$

Wir schreiben auch: $L_1 \circ L_2 = L_1 L_2$

Beispiel:
$L_1 = \{\varepsilon, ab, abb\}$
$L_2 = \{b, ba\}$
$L_1 \circ L_2 = \{b, abb, abbb, ba, abba, abbba\}$
$L_2 \circ L_1 = \{b, ba, bab, baab, babb, baabb\}$

n-te Potenz von L
$L^0 = \{\varepsilon\}$
$L^{n+1} = L^n \circ L$
$L = \{a, ab\}$
$L^0 = \{\varepsilon\}$
$L^1 = L^0 \circ L = \{\varepsilon\} \circ L = L$
$L^2 = L^1 \circ L = L \circ L = L^2 = \{a,ab\} \circ \{a,ab\} = \{aa, aab, aba, abab\}$
$L^3 = L^2 \circ L = \{aa, aab, aba, abab\} \circ \{a,ab\} = \{aaa, aaba, abaa, ababa, aaab, aabab, abaab, ababab\}$

Definition: Kleene-Stern-Produkte
$$L^* = \bigcup_{n \geq 0} L^n = L^0 \cup L^1$$
Beispiel: Spiegelung (sp)
$$sp : \Sigma^* \to \Sigma^*$$
$$sp(\varepsilon) = \varepsilon$$
$$a \in \Sigma, w \in \Sigma^* \quad sp(wa) = a \circ sp(w)$$
Für Sprachen (SP)
$$SP : P(\Sigma^*) \to P(\Sigma^*)$$
$$SP(L) = \{sp(w) \mid w \in L\}$$



Beschreibung der Zustände und Zustandsübergänge:
Zustandsspeicher:
- eine einzige Speicherzelle (!)
- Inhalt: aktueller Zustand des Automaten

Zustandsmenge:
- Beschreibung der endlich vielen möglichen Zustände

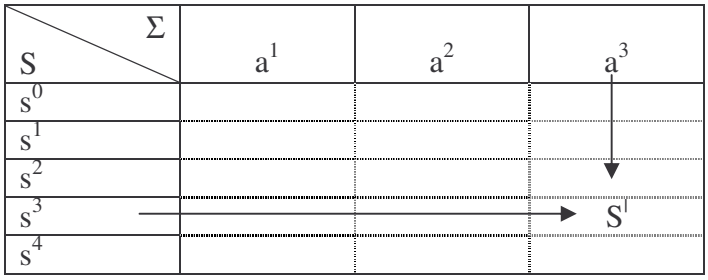
Beispiel anhand unseres Schwimmbadautomaten A_s:
$S = \{start, S_{50}, S_{100}, S_{150}, S_{200}\}$ (S = Zustandsspeicher)
Startzustände: $S_0 \in S$ nur einer !
Endzustände: $F \subseteq S$ mehrere !

Der Automat akzeptiert die Eingabefolge, falls er sich – beginnend mit dem Startzustand – nach Bearbeitung der kompletten Eingabefolge in einem Endzustand befindet.
A_s : Startzustand: start
Endzustände: $F = \{S_{200}\}$

Zustandsüberführung:

δ : S x Σ à S

$s \in S, a \in \Sigma \quad \delta(s,a) = s^1 \quad s^1 \in S$



Definition:

Ein deterministischer, endlicher Automat ist ein System aus A = (Σ, S, δ, s_0, F). Dabei ist…

1. Σ = das Eingabealphabet
2. S = die Zustandsmenge
3. δ = S x Σ à S (die Zustandsüberführungsfunktion, S verarbeitet Σ)
4. $s_0 \in$ S (Startzustand)
5. F $\subseteq$ S (Menge der Endzustände)

1. Bemerkung:

δ muss keine totale Funktion sein, d.h. es ist nicht notwendigerweise für alle Paare definiert (z.B.: (s,a) ∈ S x Σ

Beispiel:

A_s : δ (S_{200}, 50) ist nicht definiert

2. Bemerkung:

A_s heißt deterministisch, weil zu jedem Zustand und jedem Eingabesymbol höchstens ein Folgezustand existiert (δ ist eine Funktion).

Beispiel:

A_s = {50, 100, 200}, {start, s_{50}, s_{100}, s_{150}, s_{200}}, δ, start, {s_{200}}

δ (start, 50) = S_{50}, δ (start, 100) = S_{100},…

Endlicher Automat A als Programm auffassen:

Mögliche Eingaben: Σ*

Verarbeitungszustände: s

Programmstart: s_0

Programmende: s_{200}

Anweisungen: δ

Andere Darstellung für Anweisungen:

δ {(s,a,s') | δ (s,a) = s'}

Sei A = (Σ, S, δ, s_0 F) ein endlicher Automat

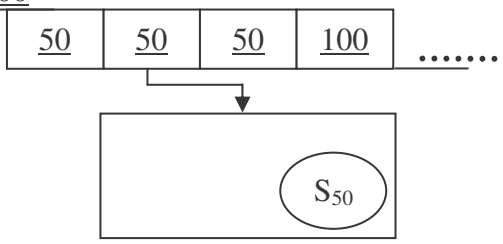
δ erweitern zu δ*

δ*: S x Σ* à S

δ*: (S, ε) = S

$a \in \Sigma, w \in \Sigma^* \qquad \delta^* : (S, aw) = \delta^*(\delta(s,a), w) = \delta^*(\delta(s', w))$

Beispiel: As, s=s50, v= 50 50 50

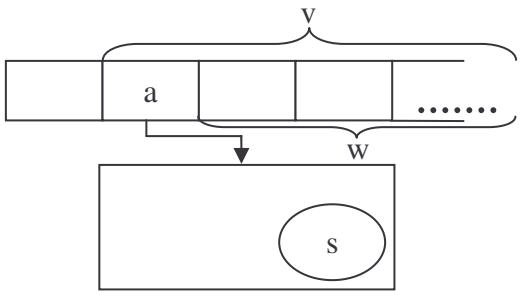


$\delta^*: (s_{50}, \underline{50}\ \underline{50}\ \underline{100}) = \delta^*: (\delta\ (s50, \underline{50}), \underline{50}\ \underline{100}) = \delta^*: (s_{100}, \underline{50}\ \underline{100}) = \delta^*: (s_{150}, \underline{100}) = \delta^*: (s_{200}, \varepsilon) = s_{200}$

Konfiguration (k):

k = (s, v) $\quad s \in S, v \in \Sigma^*$ (s = Zustand, v = Restwort)

wobei s der aktuelle Zustand ist und v das noch zu verarbeitende Suffix des Eingabewortes



Menge der Konfigurationen: K = S x Σ*

Menge aller Konfigurationen K:

$K = \{(s,v) \mid s \in S, v \in \Sigma^*\} S x \Sigma^*$

Folgekonfigurationen:

k = (s,v) $\quad$ v = a°w $\quad$ (a ∈ ∑, w ∈ $\sum^*$)

= (s,aw) = δ (s,a) ⇒ k' = (s', w)

$k \mapsto k'$ (k' ist die Folgekonfiguration von k)

Beispiel mit unserem Schwimmbadautomat:

$\delta^* = (start, \underline{50}\ \underline{100}\ \underline{100})$

$\mapsto (s_{50}, \underline{100}\ \underline{100})$

$\mapsto (s_{150}, \underline{100})$

$\mapsto (s_{200}, \varepsilon)$

$(s_0, x_1\ x_2 \ldots x_r)$

$\mapsto (s_1, x_2 \ldots x_r)$

$\mapsto$ .....

$\mapsto (s_r, \varepsilon)$

wobei $\delta(s_i, x_{i+1}) = s_{i+1}$

falls s_0 der Startzustand und s_r der Endzustand sind, dann wird $x = x_1 \ldots x_r$ akzeptiert.

$\mapsto \subseteq KxK$

$\subseteq (Sx\Sigma^*)x(Sx\Sigma^*)$

Bezeichnung:

$\mapsto^*$ bezeichnet die reflexive-transitive Hülle der Relation $\mapsto$; rekursiv definiert durch:

1. $k \mapsto^* k$ (eine Konfiguration steht zu sich selbst in Relation) für alle $\in K$
2. $k \mapsto^* k'$ falls ein weiteres k'' ∈ K existiert mit $k \mapsto k''$ und $k'' \mapsto^* k'$

   Mit anderen Worten:

   Wenn es eine Folge von Konfigurationsübergängen gibt $k_1 \mapsto k_2 \mapsto k_3 \mapsto \ldots \mapsto k_n$, dann gilt

   $k_1 \mapsto^* k_n$

   $A_s = (start, \underline{50}\,\underline{100}\,\underline{100}) \mapsto^* (s_{200}, \varepsilon)$

   δ = funktional, $\mapsto$ = operational

Reguläre Sprachen

Definition:

a) $A = (\Sigma, S, \delta, s_0, F)$ sei ein endlicher Automat $L_{(A)} = \{w \in \Sigma^* \mid (s_0, w) \mapsto^* (s, \varepsilon), s \in F\}$

Die Sprache $L_{(A)}$ heißt „die von A akzeptierte Sprache

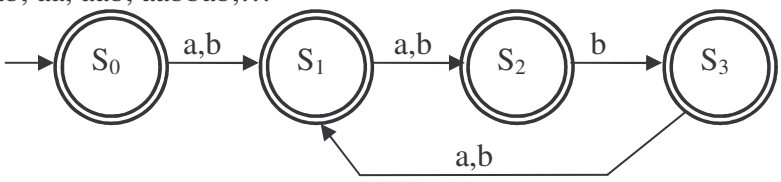
b) Eine Sprache $L \subseteq \Sigma^*$ heißt regulär, falls es einen endlichen Automaten A gibt mit $L_{(A)} = L$

Bezeichnung:

DFA_Σ ist die Klasse aller von endlichen Automaten akzeptierten Sprachen über das Alphabet Σ

$DFA_\Sigma = REG_\Sigma$

Definition:
Zwei endliche Automaten A und A' heißen äquivalent, falls $L_{(A)} = L_{(A')}$
Beispiel:
$L_{3b} = \{w \in \{a,b\}^* \mid w = w_1 ... w_n \mid w_i \in \{a,b\}, w_{3i} \neq a,\ i \geq a,\ n \geq 0\}$
wenn i durch drei teilbar ist, darf das dritte Zeichen kein a sein.
Beispielwörter: a, b, ab, aa, aab, aabbab,…

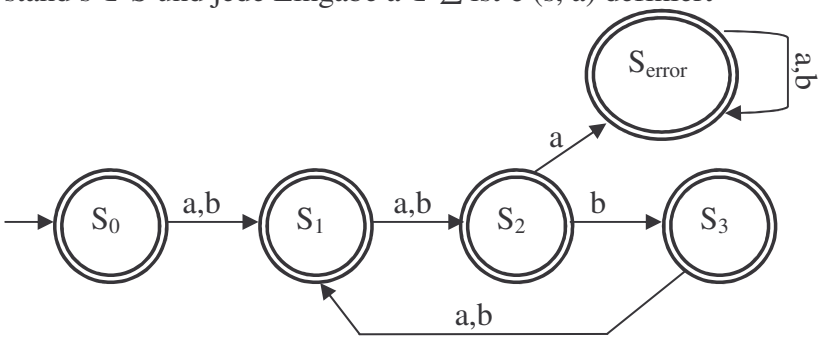


$A_{3b} = (\{ab\}, \{s_0, s_1, s_2, s_3\}\delta, s_0, \{s_0, s_1, s_2, s_3\})$

$L(A_{3b}) = L_{3b}$ $\quad$ $\delta(s_1, b) = s_2$
$\delta(s_0, a) = s_1$ $\quad$ $\delta(s_2, b) = s_3$
$\delta(s_0, b) = s_1$ $\quad$ $\delta(s_3, a) = s_1$
$\delta(s_1, a) = s_2$ $\quad$ $\delta(s_3, b) = s_1$

Vollständiger Automat:

δ musste nicht total definiert sein.
Definition:
Ein endlicher Automat $A = (\Sigma, S, \delta, s_0, F)$ heißt vollständig, falls δ total definiert ist, d.h. für jeden Zustand s ∈ S und jede Eingabe a ∈ ∑ ist δ (s, a) definiert



$A_{3b}^{total} = (\{a,b\}, \{s_0, s_1, s_2, s_3, s_{error}\}, \delta_{total}, s_0, \{s_0, s_1, s_2, s_3\})$

$$\delta_{total}(s,a) = \begin{cases} \delta(s,a), \text{ falls } \delta(s,a) \text{ definiert} \\ s_{error}, \text{ sonst} \end{cases}$$

Es gilt:
$L(A_{3b}) = L(A_{3b}^{total})$
A3b und A_{3b}^{total} sind äquivalent. Zu jedem endlichem Automaten gibt es einen äquivalenten, vollständigen Automaten.
Konstruktion des Automaten:
A = (Σ, S, δ, s_0, F) ist ein beliebiger, endlicher Automat. Wir konstruieren einen äquivalenten, vollständigen Automaten:

$s_{error} \notin S;$

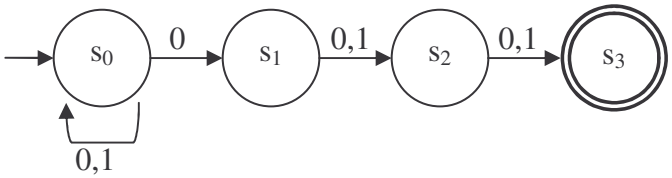
$A_{total} = (\Sigma, S \cup \{s_{error}\}, \delta_{total}, s_0, F)$

$$\delta_{total}(s,a) = \begin{cases} \delta(s,a) \; falls \;\; \delta(s,a) definiert \\ s_{error}, sonst \end{cases}$$

$a \in \Sigma; \delta_{total}(s_{error}, a) = s_{error}$ diese Art Übergänge sind per Konstruktion schon mit dabei.

Nicht-deterministische, endliche Automaten

Beispiel: $L_1 = \{w \in \{0,1\}^* \mid w = u0v, u \in \{0,1\}^*, v \in \{00,01,10,11\}\}$



$\delta(s_0,0)$?

Definition:

Ein nicht deterministischer, endlicher Automat ist ein System $A = (\Sigma, S, \delta, s_0, F)$

Σ = Eingabealphabet
S = Zustandsmenge
$s_0 \subseteq S$ = Menge von Startzuständen
$F \subseteq S$ = Menge von Endzuständen
$\delta \leq S \times \Sigma \times S$ = die Überführungsrelation
δ ist eine Menge von Tripeln (s_i, a, s_j)

$A_1 = (\{0,1\}, \{s_0, s_1, s_2, s_3\}, \delta, \{s_0\}, \{s_3\})$

mit $\delta = \{(s_0,0,s_0), (s_0,0,s_1), (s_0,1,s_0), (s_1,0,s_2), (s_1,1,s_2), (s_2,0,s_3), (s_2,1,s_3)\}$

δ	s_0	s_1	s_2	s_3
0	$\{s_0,s_1\}$	$\{s_2\}$	$\{s_3\}$	$\{\}$
1	$\{s_0\}$	$\{s_2\}$	$\{s_3\}$	$\{\}$

$\delta = S \times \Sigma$ à P (S)

$\delta = (s_0,0) = \{s_0, s_1\}$

$\delta = (s_0,1) = \{s_0, s_1\}$

.....

$\delta = (s_3,0) = \{\{\}\}$

Konfiguration k = (s,v)

Übergang:

$k = (s, aw) \mapsto (s', w), falls$

$\delta(s,a) = \{s'\}$

$(s,a,s') \in \delta$

Die von einem nicht-deterministischem Automaten akzeptierte Sprache:

$L(A) = \{w \in \Sigma^* \mid (s_0, w) \mapsto^* (s, \varepsilon), s_0 \in S, s \in F\}$

Bezeichnung:

NFA_Σ = Klasse der von nicht-deterministischen, endlichen Automaten akzeptierten Sprachen über Σ,

Beispiel: A_1 $1100011 \in L(A)$?

Nach Definition ist zu prüfen:

$(s_0,1100011) \mapsto *(s_3,\varepsilon)?$

$(s_0,1100011) \mapsto (s_0,100011) \mapsto (s_0,00011)$

$\mapsto (s_0,0011) \mapsto (s_0,011)^{*1} \mapsto (s_0,11)$

$\mapsto (s_0,1) \mapsto (s_0,\varepsilon)$

$Bei\ ^{*1} \mapsto (s_1,11) \mapsto (s_2,1) \mapsto (s_3,\varepsilon) \leftarrow (Back-tracking)$

Um zu zeigen, dass eine Eingabe w von einem nicht-deterministischem Automat akzeptiert wird, gilt es, eine Konfiguration zu finden.
Das Eingabewort wird abgearbeitet und am Endzustand errechnet. Nur wenn alle Konfigurationsfolgen, bei denen das Wort abgearbeitet wird, nicht in einem Endzustand enden, dann wird das Wort nicht akzeptiert.

$\delta^* = P(S) \times \Sigma^* \rightarrow P(S)$

$\delta^* = (R, \varepsilon) \rightarrow R \mid R \in P(S)$

$\delta^* = (R, aw) \rightarrow \delta^*\left(\bigcup_{s \in R} \delta(s,a), w\right)$ $\left(\bigcup_{s \in R} = Vergrößerungsmenge\ \ mit\ s \in R\right)$

$\delta^* = (s_0, w) = R \qquad R \cap F \neq \{\}$, *dann wird w akzeptiert*

Beispiel: w = 1100011

$\delta^* = (\{s_0\}w) = (\{s_0\}, 1100011)$

$\delta^*(\{s_0\}100011) = \delta^*(\{s_0\Delta\}00011) = \delta^*(\{s_0, s_1\}0011)...$

$\delta^* = \bigcup_{s \in \{s0, s1\}} \delta(s,0), 011 = \delta^*(\delta(s_0,0) \cup \delta(s_1,0), 011) = \delta^*(\{s_0, s_1\} \cup s_2), 011 = \delta^*(\{s_0, s_1, s_2\}, 011)$

$= \delta^*(\{s_0, s_1, s_2, s_3\}, 11) = \delta^*(\{s_0, s_2, s_3\}, 1) = \delta^*(\{s_0, s_3\}, \varepsilon)$

Allgemein gilt:

$L_{(A)} = \{w \in \Sigma^*, \delta^*(s_0, w) \cap F \neq \{\}\}$

Fragestellungen:

$DFA_\Sigma \overset{?}{=} NFA$

$DFA_\Sigma \overset{?}{\subseteq} NFA$

Gegeben sei ein deterministischer, endlicher Automat A = (Σ, S, δ, s_0, F) sowie ein nicht-deterministischer, endlicher Automat A_{nd} = (Σ, S, δ_{nd}, {s_0}, F)

$a \in \Sigma \quad (s, a, s') \in \delta_{nd}$, genau dann wenn $\delta(s,a) = s'$

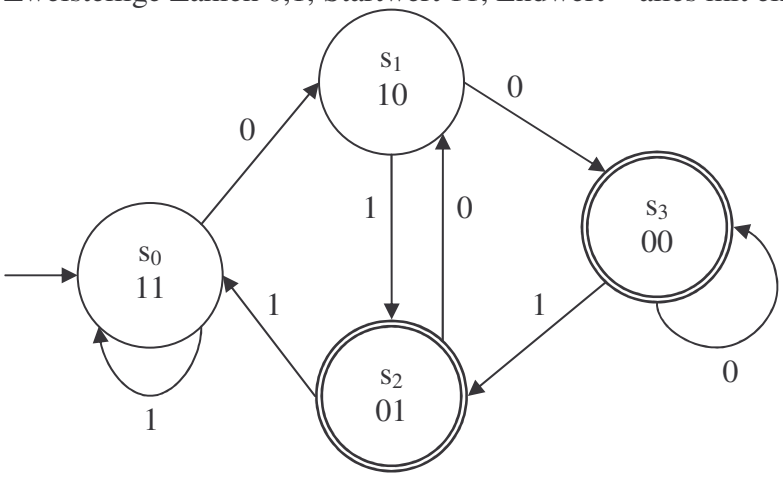
Es gilt $L_{(A)} = L_{(And)}$, daher $DFA_\Sigma \subseteq NFA_\Sigma$

$NFA_\Sigma \subseteq DFA_\Sigma$?

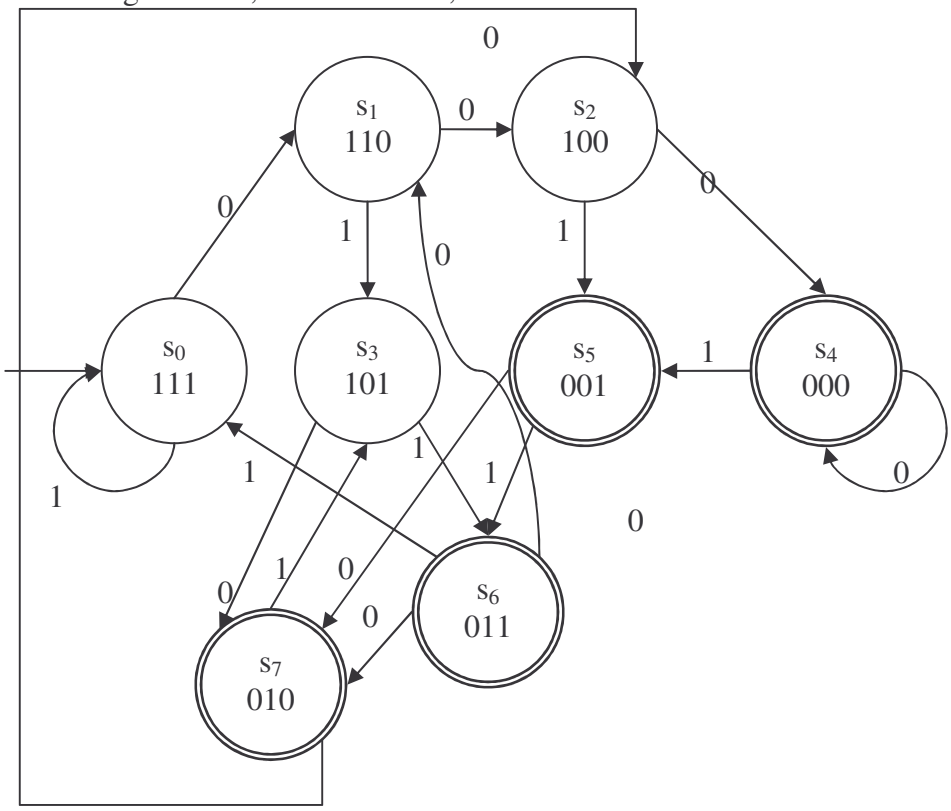
$\Sigma = \{0,1\}$

L1 mit deterministischem Automat beschreiben

a) Zweistellige Zahlen 0,1, Startwert 11, Endwert = alles mit einer 0 vorne:



b) Dreistellige Zahlen, Startwert: 111, Endwert = alles mit einer 0 vorne



$$A_1^{\ d} = (\{0,1\}, \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}, \delta, s_0, \{s_4, s_5, s_6, s_7\})$$

Algorithmus zur Konstruktion:

Gegeben: Ein nicht-deterministischer Automat A = (Σ, S, δ, s_0, F), δ = S x Σ à P(S)
Konstruiere einen deterministischen Automaten A_d mit $L_{(A)} = L_{(Ad)}$.
Zustandsmenge an Ad: P(S)
Startzustand: S_0 (Menge der Startzustände von A
Menge der Endzustände: $F_d := \{R \subseteq S \mid R \cap F \neq \{\}\}$

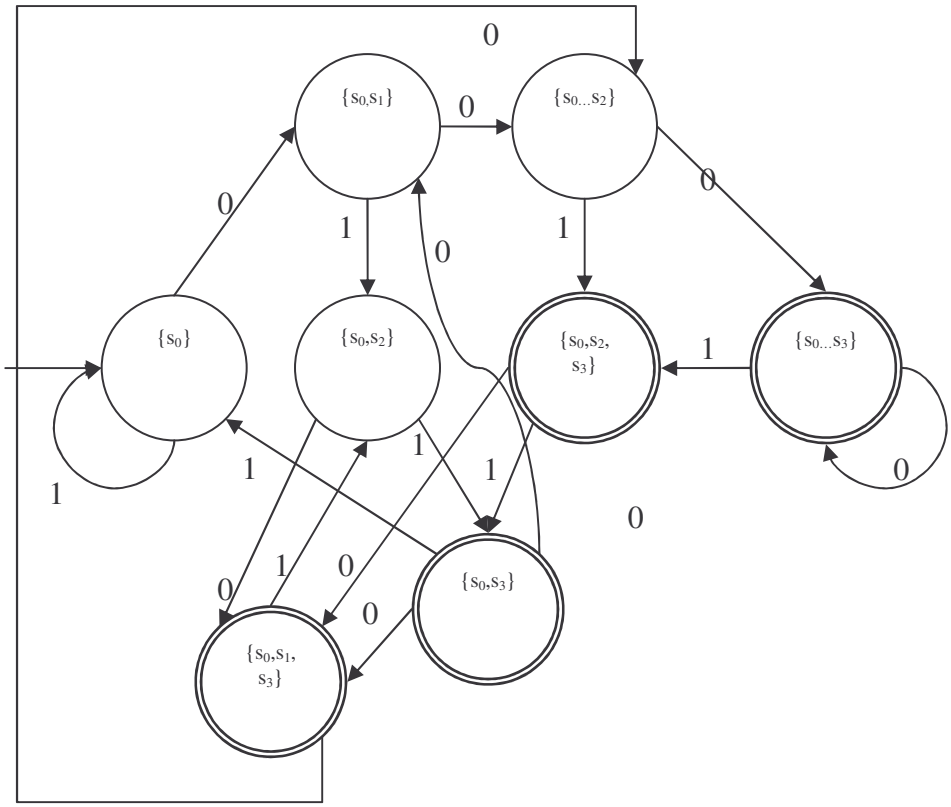
$R \subseteq S, a \in \Sigma$

$$\delta_d(R,a) = \underset{S \in R}{U} \delta(s,a)$$

$$A_d = \left(\Sigma, P(S), \delta_d, s_0^{\ s} = S_0 = F_d\right)$$

Beispiel: Wir verwenden die Konstruktion auf A_1 an L_x nach unserem oben abgebildetem Automaten:

1. $\delta_d(\{s_0\},0) = \{s_0, s_s\}$
   $\delta_d(\{s_0\},1) = \{s_0\}$
2. $\delta_d(\{s_0, s_1\},0) = \{s_0, s_s, s_2\}$
   $\delta_d(\{s_0, s_1\},1) = \{s_0, s_2\}$
3. $\delta_d(\{s_0, s_1, s_2\},0) = \{s_0, s_s, s_2, s_3\}$
   $\delta_d(\{s_0, s_1, s_2\},1) = \{s_0, s_2, s_3\}$
4. $\delta_d(\{s_0, s_2\},0) = \{s_0, s_{1s}, s_3\}$
   $\delta_d(\{s_{01}, s_2\},1) = \{s_0, s_3\}$
5. $\delta_d(\{s_0, s_1, s_2, s_3\},0) = \{s_0, s_s, s_2, s_3\}$
   $\delta_d(\{s_0, s_1, s_2, s_3\},1) = \{s_0, s_2, s_3\}$
6. $\delta_d(\{s_0, s_2, s_3\},0) = \{s_0, s_s, s_3\}$
   $\delta_d(\{s_0, s_2, s_3\},1) = \{s_0, s_3\}$
7. $\delta_d(\{s_0, s_1, s_3\},0) = \{s_0, s_s, s_2\}$
   $\delta_d(\{s_0, s_1, s_3\},1) = \{s_0, s_2\}$
8. $\delta_d(\{s_0, s_3\},0) = \{s_0, s_1\}$
   $\delta_d(\{s_0, s_3\},1) = \{s_0\}$



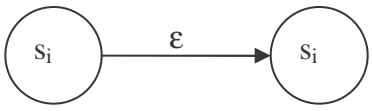
$s_0{}^d\{s_0\}$

δ^d

$F^d = \{\{s_0, s_3\}, \{s_0, s_1, s_3\}, \{s_0, s_2, s_3\}, \{s_0, s_1, s_2, s_3\}\}$

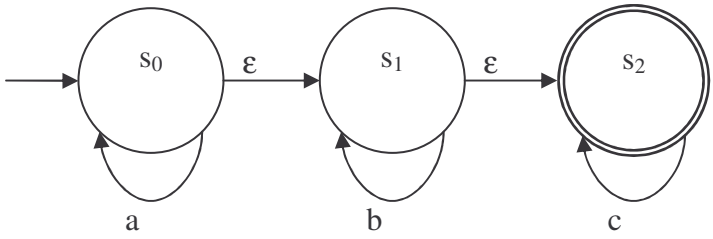
$A^d\left(\Sigma, P(S), \delta^d, s_0{}^d, F^d\right)$

Endliche Automaten mit ε-Übergängen



Beispiel:

$L_{abc} = \{a^i, b^j, c^k \mid i, j, k \geq 0\}$



$A_{abc} = (\{a, b, c\}, \{s_0, s_1, s_2\}, \delta, \{s_0\}, \{s_2\})$

Definition:

Ein endlicher ε-Automat ist definiert durch A = (Σ, S, δ, s_0, F)

Σ = Eingabealphabet
S = endliche Menge von Zuständen
S_0 = Menge von Startzuständen
F = Menge von Endzuständen
δ = Zustandsüberführungsfunktion ($\delta \subseteq S \times (\Sigma \cup \{\varepsilon\}) \times S$)
Mögliche Übergänge: $(s, a, s'), (s, \varepsilon, s')$

Konfiguration:

$k = (s, w) \quad w \in \Sigma^*$

$k \mapsto k'$

$(s,aw) \mapsto (s',w)$ genau dann, wenn $(s,a,s') \in \delta \quad a \in \Sigma \cup \{\varepsilon\}$

Sprache eines ε-Automaten:

$L_{(1)} = \{w \in \Sigma^* \mid (s_0, w) \mapsto^* (s, \varepsilon), s_0 \in S_0, s \in F\}$

Beispiel:

εFA_Σ = Klasse der Sprachen über Σ, die von einem ε-Automaten akzeptiert werden

Satz:

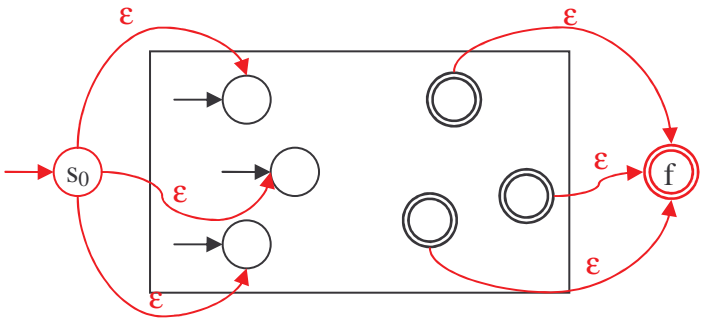
$\varepsilon FA_\Sigma = NFA_\Sigma (= DFA_\Sigma)$

$\supseteq$ gilt, da alle Übergänge, die ein nicht-deterministischer Automat erlaubt, auch mit ε-Automaten möglich sind. „$\subseteq$“ Elimination der ε-Übergänge.

Algorithmus zur Elimination der ε-Übergänge:

Sei A = (Σ, S, δ, s_0, F) ein beliebiger ε-Automat gegeben.

1. Transfpрmiere A in einen Automaten A_1 mit genau einem Startzustand s_0 und einem Endzustand f

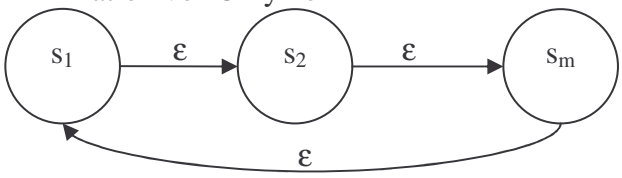


   $A_1 = (\Sigma, S_1, \delta_1, s_0, F)$

   $S_1 = S \cup \{s_0, f\}$, *wobei* $s_0, f \notin S$

   $\delta_1 = \delta \cup \{(s_0, \varepsilon, s) \mid s \in S\} \cup \{(s, \varepsilon, f) \mid s \in F_1\}$

2. Elimination von ε-Zyklen

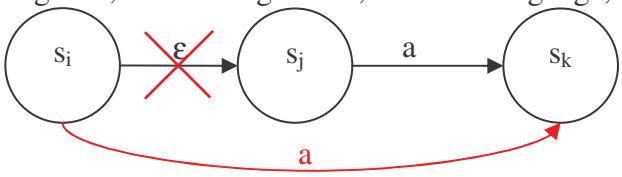


   ε-Zyklus: Folge von Zuständen $s_1, s_2, \ldots, s_n, s_m$ mit:

   $(s_1\ \varepsilon\ s_s) \in \delta, (s_n\ \varepsilon\ s_{,}) \in \delta, \ldots, (s_n\ \varepsilon\ s_m) \in \delta$

   s_1 und s_m werden aus der Zustandsmenge S herausgenommen und durch einen Zustand s_ε ersetzt. Alle Übergänge von oder zu einem der Übergänge s_1 und s_m werden jetzt auf s_ε gerichtet à A_2

3. Ergänze, wie im folgendem, solche Übergänge, bis keine neuen mehr hinzukommen können à A_3

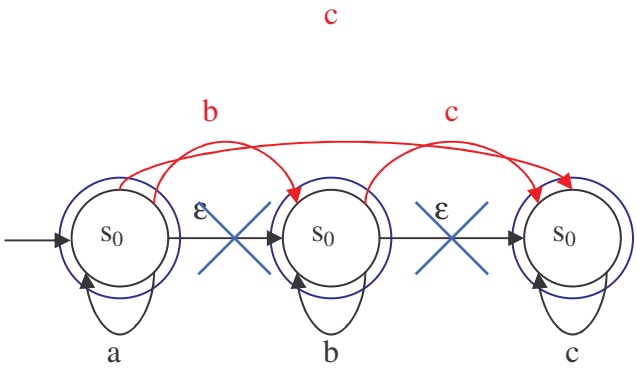


4. Endzustandsmenge $F' := \{t \in S_3 \mid (t, \varepsilon) \mapsto^* (f, \varepsilon)\}$, dann kann f eliminiert werden und es können alle ε-Übergänge eliminiert werden à A_4

Es gilt: $L(A_1) = L(A_2) = L(A_3) = L(A_4)$

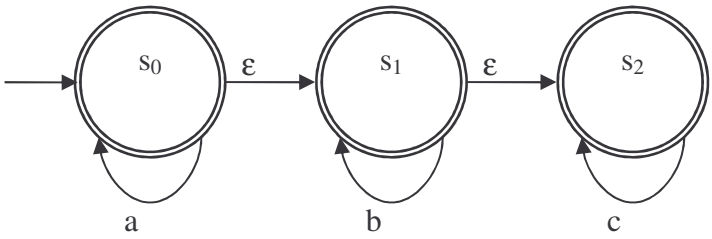
Satz: $DFA_\Sigma = NFA_\Sigma = \varepsilon FA_\Sigma$

Beispiel hierfür noch:

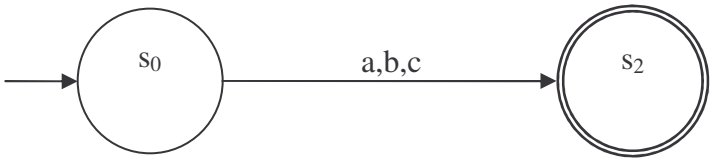


Schritt 1 und 2 unnötig
rot = Schritt 3
blau = Schritt 4

$L_{abc} = \{a^i \mid i \geq 0\} \circ \{b^j \mid j \geq 0\} \circ \{c^k \mid k \geq 0\}$



Verallgemeinerte, endliche Automaten



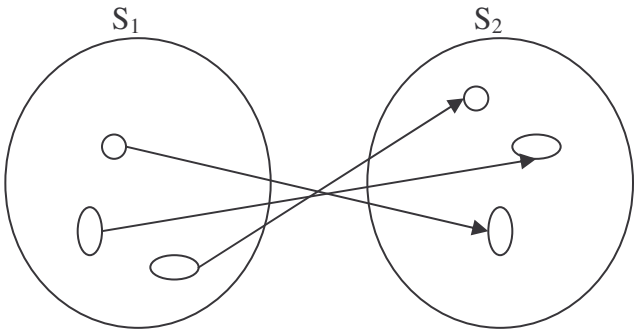
$GFA_\Sigma = DFA_\Sigma = NFA_\Sigma = \varepsilon FA_\Sigma$

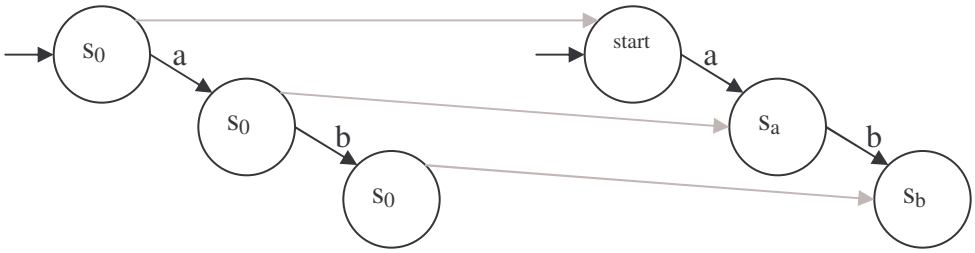
Minimierung von deterministischen, endlichen Automaten

Definition:
L sei eine reguläre Sprache über Σ; A sei ein deterministischer, endlicher Automat über Σ, der L akzeptiert: L(A) = L
A heißt minimaler Automat, falls es keinen Automaten A' mit weniger Zuständen gibt, der L akzeptiert.
Zu jeder Sprache L gibt es einen minimalen Automaten und der minimale Automat ist eindeutig bis auf die Bezeichnung der Zustände.
Äquivalenz: A_1, A_2 sind äquivalent, wenn zwei Automaten die gleiche Sprache beschreiben ($L(A_1) = L(A_2)$)

Isomorphie von endlichen Automaten

Zwei Automaten sind isomorph, wenn sie „gleich" sind, bis auf die Bezeichnung der Zustände.
Definition
$A_1 = (\Sigma_1, S_1, \delta_1, s_0, F_1)$ und $A_2 = (\Sigma_2, S_2, \delta_1, s_0, F_2)$ seien endliche Automaten über Σ mit $|S_1| = |S_2|$ (gleiche Anzahl von Zuständen).
A_1 und A_2 heißen isomorph, wenn es eine bijektive Abbildung f:=S_1 à S_2 gibt = „Umbenennung" mit
(*) $f(\delta_1(s,a)) \Rightarrow s' \Leftarrow (\delta_2(f(s),a))$ für alle $s \in S$, $a \in \Sigma$ und (**)
$f(F1) = F2$
Wir schreiben: $A_1 \cong A_2$





$$\begin{array}{llll} A_1 & S_1 & x\Sigma \xrightarrow{\delta_1} & S_1 \\ & f \downarrow & /// & f \downarrow \leftarrow \textit{Umgenennung} \text{ , /// = Diagramm kommudiert} \\ A_2 & S_2 & x\Sigma \xrightarrow{\delta_1} & S_2 \end{array}$$

Algorithmus zum Erzeugen eines minimalen Automaten

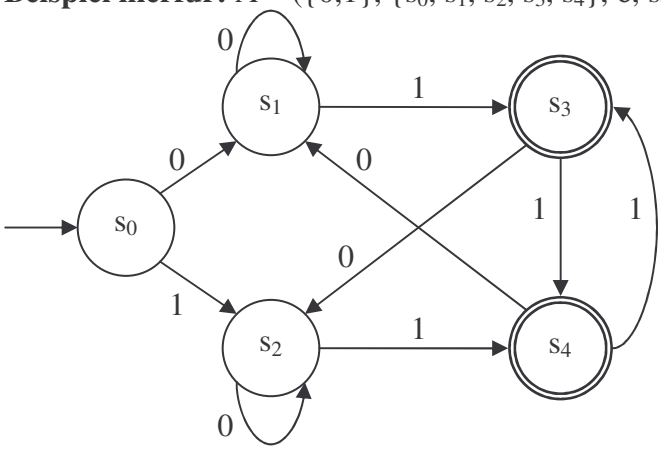
Gegeben: Ein deterministischer, endlicher Automat $A = (\Sigma, S, \delta, s_0, F)$
Wir konstruieren schrittweise einen minimalen Automat A_{min} mit $L(A) = L(A_{min})$
$\pi_i = (S_{i1},\ldots,S_{ik1})$
S_{ij} sind Teilmengen von S
π_i ist eine Zerlegung von S, übergehen zu π_{i+1}, eine neue Zerlegung, die höchstens „feiner" ist als π_i.

1. Schritt $\pi_1 = (S_{11}, S_{12})$
$S_{11} = F$ (Menge der Endzustände)
$S_{12} = S - F$ (alle übrigen Zustände)

2. Schritt $\pi_i = (S_{11}, \ldots, S_{iki})$
Wir erzeugen π_{i+1}:
s,s' aus einem Block in π_i gehören auch in π_{i+1} zu einem Block, falls für alle möglichen Eingaben ($a \in \Sigma$, $s,s' \in S$) $\delta(s,a)$ und $\delta(s',a)$ in einem Block liegen

3. Schritt
- Falls $\pi_{i+1} = \pi_i$, dann ist π_1 die Zustandsmenge von A_{min}
- Falls $\pi_{i+1} \neq \pi_i$, so wird der Algorithmus mit π_{i+1} im Schritt 2 fortgesetzt

Beispiel hierfür: $A = (\{0,1\}, \{s_0, s_1, s_2, s_3, s_4\}, \delta, s_0, \{s_3, s_4\})$

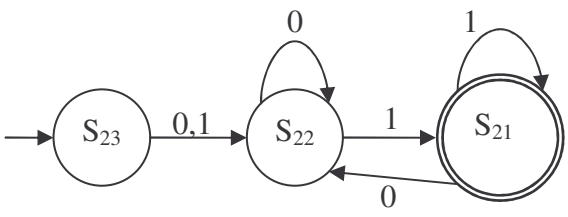


1. Schritt: $\pi_1 = (S_{11}, S_{12}) = (\{s_3, s_4\}, \{s_0, s_1, s_2\})$
2. Schritt: Zustandstabelle
Wenn eine 0 oder eine 1 folgt, kommt man von welchem Zustand in welche Zustandsmenge?

	S_{11}		S_{12}		
	s_3	s_4	s_0	s_1	s_2
0	S_{12}	S_{12}	S_{12}	S_{12}	S_{12}
1	S_{11}	S_{11}	S_{12}	S_{11}	S_{11}

$\Rightarrow \pi_2 = (S_{21}, S_{22}, S_{23})$
$S_{21} = \{s_3, s_4\}$, $S_{22} = \{s_1, s_2\}$ $S_{23} = \{s_0\}$

	S_{21}		S_{22}		S_{23}
	s_3	s_4	s_1	s_2	s_0
0	S_{22}	S_{22}	S_{22}	S_{22}	S_{22}
1	S_{21}	S_{21}	S_{21}	S_{21}	S_{22}



3. Schritt: $A_{min} = \{0,1\}, \{S_{21}, S_{22}, S_{23}\}, \delta_{min}, S_{23}, \{S_{21}\}$

Teilworterkennung

Gegeben: Irgendein Text w ($w \in \Sigma^*$)
Gesucht: Alle Stellen, an denen ein Teilstring v in W vorkommt

Pattern-Matching:

pm: $\Sigma^* \times \Sigma^* \rightarrow P(N_0)$
$w,v \in \Sigma^*$ $pm(w,v) = \{i \mid substr(w,i,i+1+|v|-1)=v\}$
|v| = Länge von v; i = Indizes, in denen v vorkommt

Beispiel:

pm = (aabaaa, aa) = {1,4,5} (An den Stellen 1, 4 und 5 des Wortes kommt aa vor)

Algorithmen für pm:

$w = w_1,\ldots,w_k$ $\quad w_i, v_j \in \Sigma \quad 1 \le i \le k$
$v_i = v_1,\ldots, v_l \quad 1 \le j \le l$

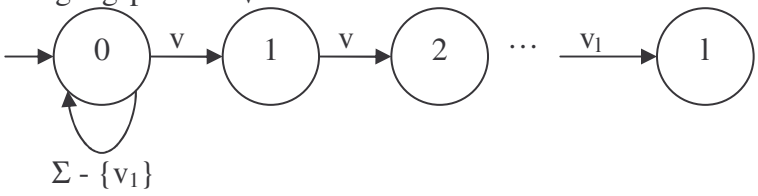
```
durchlaufe w von w1 bis wk-l+1
prüfe für jedes wi      1 ≤ i ≤ k-l+1
 ob wi … wi+l-1 = v1..vl
 falls ja: i kleinster Wert von pm(w,v)
     nein: Schleife weiter durchlaufen
 vi..vl
 w1..wi-1      wi+l-1..wk
 wi..wi+l-1
 For i = 1 to k-l+1 do
     ng:=true
     j ≤ 1
     while (j ≤ l) and ng do
       if wi+j-1 ≠ vj then ng:false
        else j:=j+1
       end if
     od
     if ng then write
(´v kommt an der Stelle ´, i, ´ vor´)
end
```

Aufwandsabschätzung

$(k - l + 1) \bullet l$
$l << k$: $(k - l + 1) \bullet l \approx k \bullet l = |w| \bullet |v|$

Effizienter mit Automaten:

Ausgangspunkt: A_v

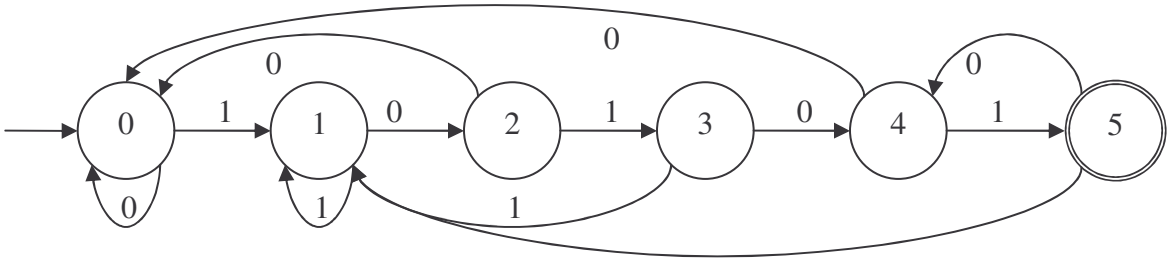


Diesen Automaten vervollständigen zu einem neuem A_v, sodass folgendes gilt:
Verarbeitet A_v die Eingabe w_ $w_1,\ldots,w_k$, dann ist A_v genau im Zustand j ($1 \le j \le l$), wenn die letzten j von A_v eingelesenen Buchstaben von w mit dem ersten j-Buchstaben von v übereinstimmen (und es keinen Index > j mit dieser Eigenschaft gibt). $w = w_1,\ldots,w_{i+1}; v_i,\ldots,v_j \mid w_{i+j}\ldots w_k$
A_v liest den nächsten Buchstaben w_{i+j}

1. $w_{i+j} = w_{j+1} \rightarrow A_v$ geht in den Zustand „j+1“ über

2. $w_{i+j} \neq w_{j+1}$ à A_v geht über in einen Zustand „R“; = kleiner als j+1, wobei R die größte Zahl ist, sodass $v_1 \ldots v_R$ Suffix von $v_1,\ldots,v_i \quad w_{i+j}$

Theoretische Informatik I



Zustand 1: nächstes Zeichen: 1

Suffix	Präfix	
1	1	ß Übereinstimmung
11	10	
111	101	ß Übereinstimmung

Zustand 2: nächstes Zeichen: 0
Zustand 3: nächstes Zeichen: 1
Zustand 4: nächstes Zeichen: 0
Zustand 5: nächstes Zeichen: 0
nächstes Zeichen: 1

Aufwand für den Algorithmus:

1. Aufstellen von A_v: $C \bullet |v|^2$; C ist eine nicht näher definierte Konstante
2. Abarbeiten von w: $|w|$

Zusammen ergibt das $C \bullet |v|^2 + |w|$ falls $v <<<< w :\approx |w|$

•	Konkatenation (Zusammenfügung)	Λ	Null-Operator
|	Selektion (Auswahl)	E	Eins-Operator
⊗	Wiederholung	[ ]	Klammern

Reguläre Ausdrücke (Syntax)

Σ = Alphabet

Die Menge der regulären Ausdrücke $R_{exp\,\Sigma}$ ist genau die Menge de Wörter über $\Sigma \cup \{\Lambda, E, \bullet, |, \otimes, [\,]\}$, die mit Hilfe folgender Regeln gebildet werden.

1. $\Lambda \in R_{exp\,\Sigma}$
2. $E \in R_{exp\,\Sigma}$
3. $a \in \Sigma = a \in R_{exp\,\Sigma}$
4. $\alpha, \beta \in R_{exp\,\Sigma}$
   a) $[\alpha \bullet \beta] \in R_{exp\,\Sigma}$
   b) $[\alpha \mid \beta] \in R_{exp\,\Sigma}$
   c) $\alpha^{\otimes} \in R_{exp\,\Sigma}$

also $\chi = \left[0 \bullet [0 \mid 1]^{\otimes}\right]$ ist regulärer Ausdruck über Σ

Beispiel: $\Sigma = \{0,1\}$ wegen der Regeln

3.: $0,1 \in R_{exp\,\Sigma}$

4.b.: $[0 \mid 1] \in R_{exp\,\Sigma}$

4.c.: $\left[0 \bullet [0 \mid 1]^{\otimes}\right] \in R_{exp\,\Sigma}$

Formale Definition von der Bedeutung eines regulären Ausdrucks (Semantik) $L : R_{exp\,\Sigma} \to P(\Sigma^*)$

1. $L(\Lambda) = \{\}$
2. $L(E) = \{\varepsilon\}$
3. $a \in \Sigma;\ L(a) = \{a\}$
4. $\alpha, \beta \in R_{exp\,\Sigma} \to L(\alpha), L(\beta)$
   a. $L([\alpha \bullet \beta]) = L(\alpha) \circ L(\beta)$
   b. $L([\alpha \mid \beta]) = L(\alpha) \cup L(\beta)$
   c. $L([\alpha^{\otimes}]) = L(\alpha)^*$ à
      $L^* = \bigcup_{n>0} L^n$ (Kleene - Stern - Produkt)

Beispiel:

$L(0) = \{0\}$

$L(1) = \{1\}$

$L([0 \mid 1]) = L(0) \cup L(1) = \{0\} \cup \{1\} = \{0,1\}$

$L([0 \mid 1]^{\otimes}) = L([0 \mid 1]^{*}) = \{0,1\}^{*}$

$L(0 \bullet [0 \mid 1]^{\otimes}) = L(0) \circ L([0 \mid 1]^{*}) = \{0\} \circ \{0,1\}^{*} = \{0\}\{0 \mid 1\}^{*}$

In der Praxis interessiert uns nur die Semantik zusammen mit der Syntax:

- $\bullet$ à $\circ$ (oder weglassen)
- E à ε
- Λ à $\{\}, \varnothing$
- $\otimes$ à $*$
- [ ] à ( )
- nur | wird nicht ersetzt

Also: $[0 \bullet [0 \mid 1]^{\otimes}]$ wird zu $(0(0 \mid 1)^{*})$

Definition:

Σ = Alphabet

$\alpha \in R_{\exp \Sigma}$

L = die Interpretation (Semantik) regulärer Ausdrücke

- Dann heißt $L(\alpha)$ die von α definierte Sprache
- Zwei reguläre Ausdrücke heißen äquivalent, falls $L(\alpha) = L(\beta)$
- $L_{R\exp \Sigma}$ = Klasse der durch reguläre Ausdrücke über Σ definierten Sprachen

Rechenregeln:

$(\alpha \mid \beta) \equiv (\beta \mid \alpha)$

$L(\alpha \mid \beta) = L(\beta \mid \alpha)$

1.) $(\varnothing \mid \alpha) \equiv \alpha \equiv (\alpha \mid \varnothing)$

2.) $(\varepsilon \bullet \alpha) \equiv \alpha \equiv (\alpha \bullet \varepsilon)$

$\{\alpha\} \circ L(\alpha) = L(\alpha)$

3.) $(\varnothing \bullet \alpha) \equiv \varnothing \equiv (\alpha \bullet \varnothing)$

$L(\varnothing) \circ L(\alpha) = L(\varnothing)$

4.) $L(\alpha^{*}) = (L(\alpha))^{*} = \bigcup_{n \geq 0} L(\alpha)^{n}$

Anwendung: Eingabefeld für Ein- und Auszahlungen auf einem Bankkonto:

- Währungskennzeichen: EUR, CHF, SEK
- Betrag:
    - Kann mit + oder – beginnen
    - Ganzzahliger Dezimalteil, durch Punkt getrennt
    - Dezimalteil genau drei Stellen
    - Ganzzahliger Anteil soll beliebig lang sein dürfen
    - Dezimalteil kann fehlen
    - falls er fehlt, wird kein Punkt verwendet

$R_{\exp \Sigma}$: Zahlung; $L(Zahlung)$ soll alle zulässigen Eingaben umfassen

Zahlung: Währungskennzeichen ° Vorzeichen ° Betrag
Währungskennzeichen = (EUR | CHF | SEK)
Vorzeichen = (+ | – | ε)
Betrag = (ganzzahliger Anteil ° (ε | (. | Dezimalteil))
Ganzzahliger Anteil = (0 | (1 .. 9) ° (0 .. 9)*)
Dezimalteil = ((0 | .. | 9) ° (0 | .. | 9))

Rechenregeln für regressive Ausdrücke:

$\alpha, \beta \in R_{\exp \Sigma}$

$(\alpha \mid \beta) = (\beta \mid \alpha)$ (α oder β) = (β oder α); Beweis :

$L(\alpha \mid \beta) = L(\alpha) \cup L(\beta) = L(\beta) \cup L(\alpha) = L(\beta \mid \alpha)$

Assoziativität:

$((\alpha \mid \beta) \mid \chi) = (\alpha \mid (\beta \mid \chi)) \equiv (\alpha \mid \beta \mid \chi)$

$((\alpha \circ \beta) \circ \chi) = (\alpha \circ (\beta \circ \chi)) \equiv (\alpha \circ \beta \circ \chi)$

Distributivgesetz:

$((\alpha \mid \beta) \circ \chi) = ((\alpha \circ \chi) \mid (\beta \circ \chi))$

$(\alpha \circ (\beta \mid \chi)) = ((\alpha \circ \beta) \mid (\alpha \circ \chi))$

$(\alpha \mid \alpha) \equiv \alpha$.

$\{\}^* = \varepsilon$.

$(\alpha \circ \alpha^*) = \alpha^+$

$(\alpha^* \circ \alpha) = \alpha^+$

α^+ = ohne leeres Wort .

$(\alpha^+ \mid \varepsilon) = \alpha^*$.

Beispiel:

^[0 – 9]{1,2}\.[0 – 9]{1,2}\.[0 - 9]{4}$

Dient zur Datumsausgabe.

Rechts einige Beispiele, die sich mit dem oben genanntem Codebeispiel ausgeben ließen:

12.12.2003
1.3.1845
17.3.1924
aber auch 99.14.9995

Metazeichen: grep (= globaly search for regular expressions and print)

.	=	Beliebiges Zeichen
\.	=	Escape à Dezimalpunkt
.+	=	Beliebige Anzahl von Zeichen, jedoch mindestens eines
.*	=	Beliebige Anzahl von Zeichen (entweder keines oder mehr als Null)
{x}	=	X-fache Wiederholung
{x,y}	=	N-fache Wiederholung, wobei n ein Wert zwischen x und y ist
[abc]	=	[a | b | c]; alternative Schreibweise: [a – z], [0 – 9], [A – Z]
^	=	Zeilenanfang
$	=	Zeilenende

Beispiel:

grep ^[aeion]*$' /usr/loca/bin/file

Perl (Practical Extraction and Report Language)

$L(R_{\exp r \Sigma}) = R_{\exp r \Sigma} (= DFA_\Sigma = NFA_\Sigma = \varepsilon FA_\Sigma)$

Satz: Zu jedem endlichem Automaten A über Σ gibt es einen regulären Ausdruck α über Σ, sodass $L(A) = L(\alpha)$

Satz: Zu jedem regulärem Ausdruck α über Σ existiert ein endlicher Automate A über Σ, sodass $L(\alpha) = L(A)$

Beweis: Konstruiere ε-Automaten

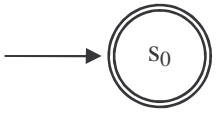
1. $\alpha = \{\}$

   $A = (\Sigma, \{s_0\}, \{\}, s_0, \{\})$

   Keine Skizzierung eines Automaten möglich

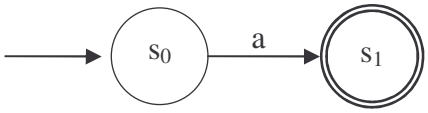
2. $\alpha = \varepsilon$

   $A = (\Sigma, \{s_0\}, \{\}, s_0, \{s_0\})$



3. $\alpha = a; a \in \Sigma$

   $A = (\Sigma, \{s_0, s_1\}, \{s_0, a, s_1\}, s_0, \{s_1\})$



4. $\beta, \chi \in R_{\text{exp }\Sigma}$

   mit B, C seien ε-Automaten mit $L(B) = L(\beta)$ und $L(C) = L(\chi)$

   $B = (\Sigma, S_B, \delta_B, s_0{}^B, \{s_f{}^B\})$

   $C = (\Sigma, S_C, \delta_C, s_0{}^C, \{s_f{}^C\})$

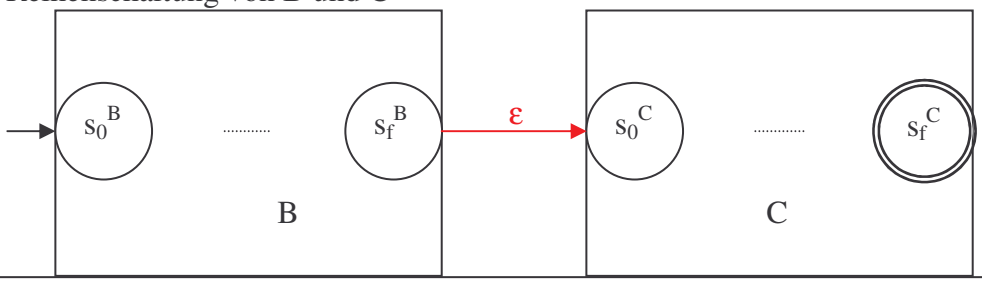
   B und C sollen nun je einen Start- und Endzustand sowie disjunkte Zustandsmengen $(S_B \cap S_C = \{\})$ besitzen

   a. $\alpha = \beta \circ \chi$

      $A = (\Sigma, S_B \cup S_C, \delta_B \cup \delta_C \cup \{s_f{}^B, \varepsilon, s_f{}^C\}, s_0{}^B, \{s_f{}^C\})$

      $L(A) = L(B) \circ L(C)$

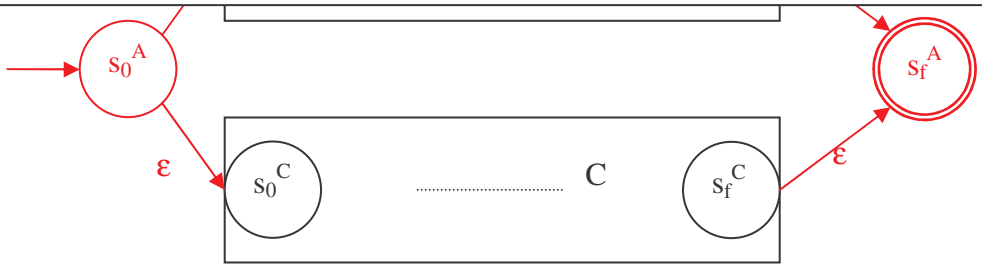
      Reihenschaltung von B und C



   $s_0{}^A, s_f{}^A \in S_B \cup S_C$

   a. $A = (\Sigma, S_B \cup S_C \cup \{s_0{}^A, s_f{}^A\}, \delta_B \cup \delta_C \cup \{(s_0{}^A, \varepsilon, s_0{}^B), (s_0{}^A, \varepsilon, s_0{}^C), (s_f{}^B, \varepsilon, s_f{}^A), (s_f{}^C, \varepsilon, s_f{}^A)\}, s_0{}^A, \{s_f{}^A$

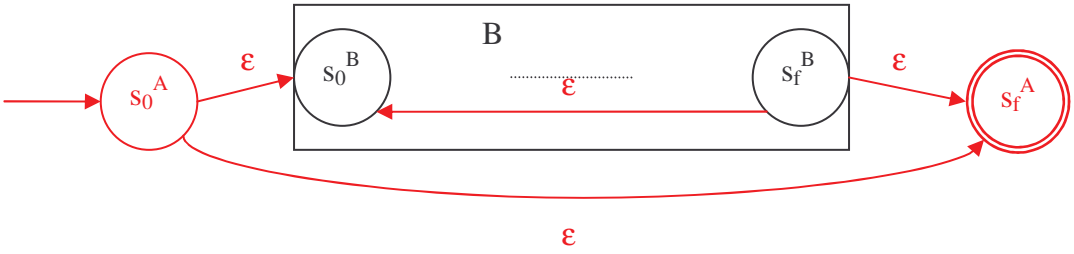
   $L(A) = L(B) \cup L(C)$



c. $\alpha = \beta^*$

$A = (\Sigma, S_B \cup S_C \cup \{s_0{}^A, s_f{}^A\}, \delta_B \cup \{(s_0{}^A, \varepsilon, s_0{}^B), (s_0{}^A, \varepsilon, s_0{}^C), (s_f{}^B, \varepsilon, s_f{}^A), (s_f{}^C, \varepsilon, s_f{}^A)\}, s_0{}^A, \{s_f{}^A\})$

$L(A) = L(B)^*$



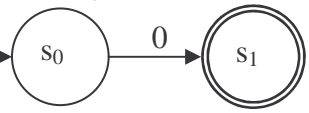
Scanner:

P,α $w \in \Sigma^* \rightarrow ja/nein; w \in L(\alpha)$

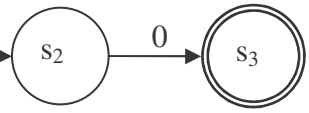
Scannergenerator: $S\varsigma \cdot \alpha \rightarrow P\alpha$

Beispiel: $\alpha = (0 \mid (0 \mid 1)^*)$

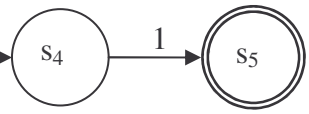
$\alpha_0 = 0$



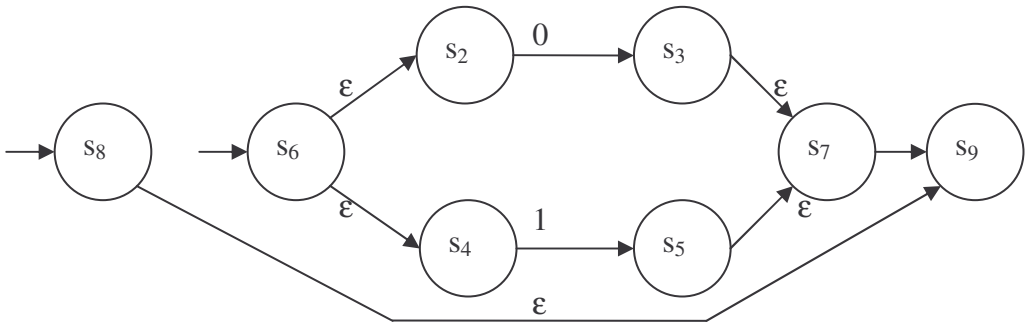
$\alpha_1 = 0$



$\alpha_2 = 1$



$\alpha_3 = (0 \mid 1)$



$\alpha_4 = (0 \mid 1)^*$

Grammatiken

Endliche Automaten: „Akzeptierende“
Reguläre Ausdrücke: „beschreibende“
Grammatik: „erzeugende“
Beispiel:
$G = (\{0,1\},\{S,A,B\},P,S)$
$P(\text{Produktionsregel}) = S \to 0S, S \to 1S, S \to 0A, A \to 0B, A \to 1B, B \to 0, B \to 1$
$x \to y$ = x darf ersetzt werden durch y; Eine Produktionsregel darf auch mehrmals durchlaufen werden
S à 01S à 010A à 0101B à $01010 \Rightarrow 01010 \in L_{(G)}$

Definition: rechtslineare Grammatiken

$G = (\Sigma, N, P, S)$
Σ = Terminalalphabet
N = Nicht-Terminalalphabet (Disjunkt von Σ)
P $\subseteq N \times (\Sigma N \cup \Sigma \cup \{\varepsilon\})$ endliche Menge von Paaren
S $\in$ N (Startsymbol)

p $\in$ P; p = (l,) $\hat{=} l \to r$ Produktion oder Regel

Ableitungsschritt
$u \in \Sigma^*, A \in N, w \in \Sigma N \cup \Sigma \cup \{\varepsilon\}$
$uA \Rightarrow uw$ genau dann, wenn $A \to w \in P$
$\Rightarrow \subseteq \Sigma^* N \times \Sigma^* (N \cup \{\varepsilon\})$
Ableitung:
Reflexive, transitive Hülle von $\Rightarrow$: $\Rightarrow^*$
$L_{(G)} = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$
Bezeichnung:
Eine Sprache L $\subseteq \Sigma^*$ heißt rechtslinear, wenn es eine rechtslineare Grammatik G über Σ gibt mit $L_{(G)} = L$. Es können folgende Arten von Regeln auftreten: $(A, B \in N, a \in \Sigma)$
1. $A \to aB$ 2. $A \to a$ 3. $A \to \varepsilon$

Vereinfachung von Grammatiken:
1. Regeln der Bauart 2 können vermieden werden
   $A \to a$
   $A \to aC \qquad C_{neu}; C = \varepsilon$
2. Nur eine bzw. zwei Regeln der Bauart 3 werden benötigt. Falls $\varepsilon \in L_{(g)}$, genügt eine Regel der Bauart 3, sonst zusätzliche: $S \to \varepsilon$

Nehmen wir an, in einer Grammatik kommen unter anderem vor:
$A_1 \to a_1 B_1 \quad A_1 = a_k T$
$A_k \to a_k B_k \quad A_k = a_k T$
} Regeln können nicht ersetzt werden
$B_1 \to \varepsilon$
$B_k \to \varepsilon$
} T_ε

Definition: linkslineare Grammatiken

Einziger Unterschied zu rechtslinearen Grammatiken: $P \subseteq N \times (N\Sigma \cup \Sigma \cup \{\varepsilon\})$

Beispiel: $G = (\{0,1\}, \{S, A, B\}, P, S)$

$P = \{S \to S0, S \to S1, S \to A0, A \to B0, A \to B1, B \to 0, B \to 1\}$

$S \Rightarrow S0 \to S00 \to S100 \to A0100 \to B10100 \to 010100$

Bezeichnungen:

LiL_Σ = Klasse der linkslinearen Sprache über Σ

ReL_Σ = Klasse der rechtslinearen Sprache über Σ

Es sei G eine rechtslineare Grammatik, dann gibt es eine linkslineare Grammatik G' mit

$L_{(G)} = SP(L(G'))$ ß = Spiegelung

Zu jeder Regel A à aB in G ersetzen wir durch A à ba in G'

Satz: Zu jeder rechtslinearen Grammatik G gibt es eine linkslineare Grammatik G' mit $L_{(G)} = L_{(G')}$ und umgekehrt, also $ReL_\Sigma = LiL_\Sigma$

Beweis:

Sei $G = (\Sigma, N, P, S)$ eine rechtslineare Grammatik, die nur Regeln der Bauart 1 (A à B0) und eine ε-Regel T à ε enthält. Evtl. zusätzlich S à ε, falls $\varepsilon \in L_{(G)}$.

Wir geben ein allgemeines Verfahren an, wie G in eine äquivalente linkslineare Grammatik verwandelt werden kann.

G = Wortaufbau von links nach rechts

G' = Wortaufbau von rechts nach links

Ergebnis sollen die gleichen Wörter (!) sein und nicht das gespiegelte.

Jede Ableitung in G endet mit T à ε

In G': T wird Startsymbol, T à ε wird eliminiert, außer für den Fall, dass $\varepsilon \in L_{(G)}$, dann behalten wir T à ε bei (d.h. es wird von hinten nach vorne das Wort aufgebaut).

Ferner wird aus A à aB in G à

B à Aa in G', da B erzeugt werden muss.

Als terminierende Regel gilt: S à ε

Beispiel (Fortsetzung)**:**

Wir konstruieren eine linkslineare Grammatik G'. C à ε wird eliminiert

$G' = (\{0,1\}, \{S, A, B, C\}, P', C)$

$P' = \{C \to B1, C \to B01, B \to A0, B \to A1, A \to S0, S \to S0, S \to S1, S \to \varepsilon\}$

Bezeichnung:

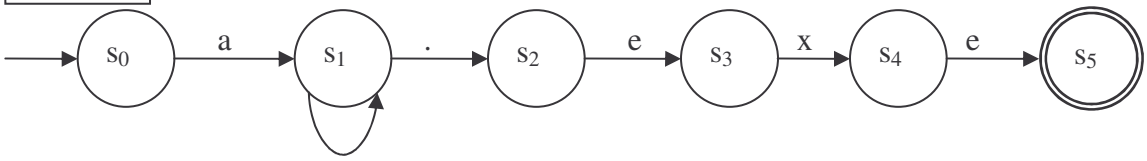
$LiL_\Sigma = ReL_\Sigma$

Typ-3-Grammatiken: rechts- oder linkslinear

Typ-3-Sprachen: $LiL_\Sigma = ReL_\Sigma = Typ3_\Sigma$

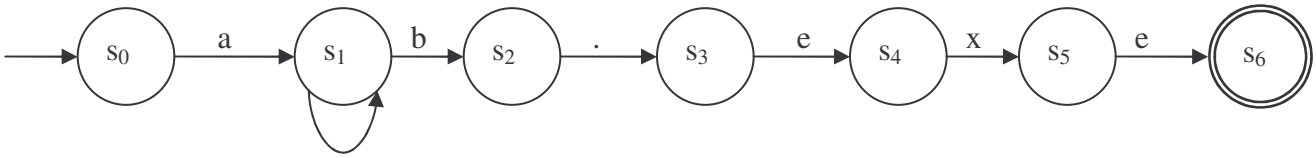
DOS: dir a*.exe
UNIX: ls a*b.exe

DOS



ASCII

UNIX



ASCII

Äquivalenz von Typ3-Grammatiken und endlichen Automaten.

Satz: Zu jeder Typ3-Grammatik G über dem Terminalalphabet Σ gibt es einen endlichen Automaten A über Σ mit $L_{(G)} = L_{(A)}$

Beweis/Algorithmus:

Sei $G = (\Sigma, N, P, S)$ eine rechtslineare Grammatik. Wir konstruieren einen endlichen Automaten

$A = (\Sigma, N \cup \{f\}, \delta, S, F); F = (\{B \in N \mid B \to \varepsilon \in P\} \cup \{f\})$

Zustandsübergänge aus den Produktionen herleiten:

1. $A \to bC \in P \Rightarrow$ ergibt $(A, b, C) \in \delta$
2. $A \to a \in P \Rightarrow$ ergibt $(A, a, f) \in \delta$
3. $B \to \varepsilon \in$ P ist schon berücksichtigt durch die Definition von f in Schritt 2.

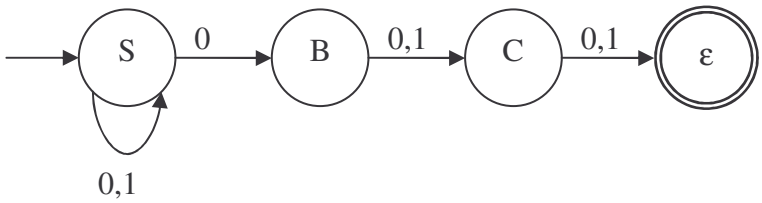
Also ergibt sich insgesamt $\delta = \{(A,b,C) \mid A \to bC \in P\} \cup \{(A,a,f) \mid A \to a \in P\}$

Beispiel:

$G = (\{0,1\}, \{S, B, C\}, P, S)$

$$P = \left\{ \begin{array}{l} S \to 0S, S \to 1S, \\ S \to 0B, B \to 0C, \\ B \to 1C, C \to 0, C \to 1 \end{array} \right\}$$

$A = (\{0,1\}, \{S, B, C, f\}, \delta, S, \{f\})$



Satz: Zu jedem endlichem Automaten A über Σ gibt es eine rechtslineare Grammatik G über Σ mit $L_{(A)} = L_{(G)}$

Beweis/Algorithmus:

$A = (\Sigma, S, \delta, s_0, F)$ sei ein endlicher Automat. Wir konstruieren $G = (\Sigma, S, P, s_0)$

1. $\delta(s,a) = s' \Rightarrow$ ergibt $s \to as' \in P$
2. $s \in F \Rightarrow$ ergibt $s \to \varepsilon \in P$

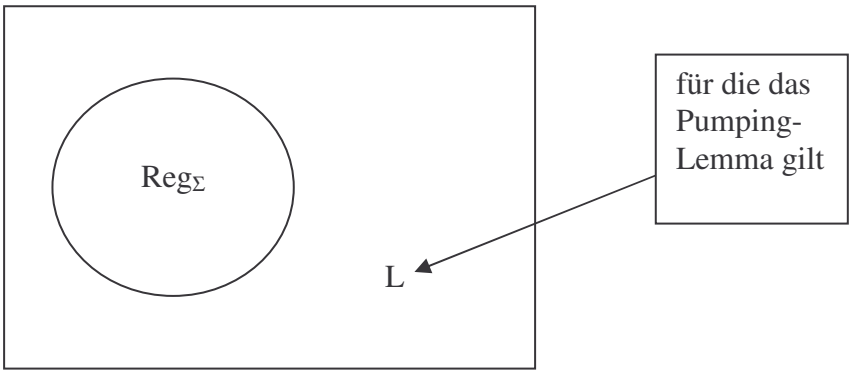
Also ergibt sich zusammen $P = \{s \to as' \mid \delta(s,a) = s'\} \cup \{s \to \varepsilon \mid s \in F\}$

$\mathrm{Re}\, g_\Sigma = TYP3_\Sigma (= DFA_\Sigma = NFA_\Sigma = \varepsilon FA_\Sigma = LR_{\exp\,\Sigma})$

Pumping Lemma für reguläre Sprachen

$\Sigma = (a,b) \quad L_1 = \{a^k b^k \mid k \geq 0\}$

Sprachen:



Wir werden zeigen, dass für L_1 das Pumping Lemma nicht gilt.

Satz: $L \in \text{Re}\, g_\Sigma$, dann gibt es ein $n \in N$, sodass sich alle $x \in L$ mit $|x| \geq m$ zerlegen lassen in:

x = uvw mit:

1. $|v| \geq 1$
2. $|uv| \leq n$
3. $uv^i w \in L$ für alle $i \in N$

Beweis: $L \in R \in G_\Sigma$

Es gibt einen endlichen Automaten A mit $L_{(A)}$ = L. n sei die Anzahl der Zustände von A.

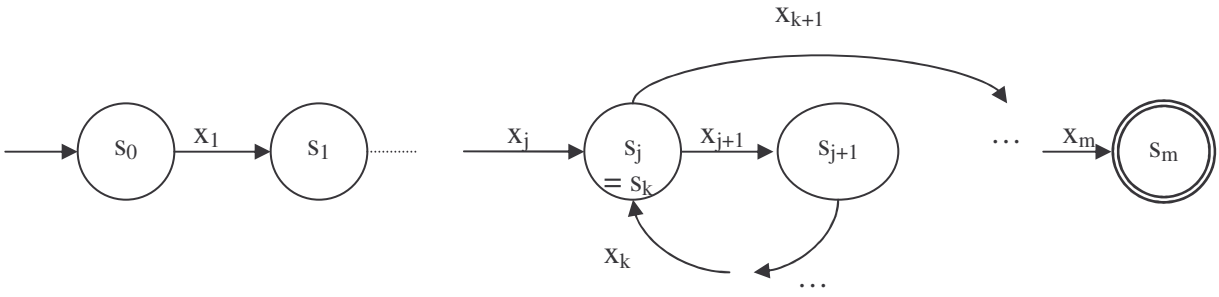
$x \in L$ mit $|x| \geq n; x = x_1 ... x_m \quad m > n \quad x_i \in \Sigma$

Da $x \in L_{(A)}$, gibt es eine Folge von Zustandsübergängen:

$\delta(s_i, x_{i+1}) = s_{i+1} \quad 0 \leq i \leq m-1$ mit s_0 = Startzustand, s_m = Endzustand.

$\underbrace{s_0, s_1, s_2, ..., s_n}_{\text{min. n+1 viele Zustände}} \leq s_m$

Es gibt ein j und ein k mit $0 \leq j < k \leq n;\ s_j = s_k$



Wähle u,v,w

$u = x_1$ bis x_j

$v = x_{j+1}$ bis x_k

$w = x_{k+1}$ bis x_m

Zu zeigen:

1. $|v| \geq 1$ gilt, da j < k
2. $|uv| \leq n$ gilt, da k < n
3. $uv^i w \in L_{(A)}$ gilt per Konstruktion

$L_1 = \{a^k b^k \mid k \geq 0\}$

Annahme: $L_1 \in \mathrm{Re}\, g_\Sigma$, dann würde L_1 das Pumping Lemma erfüllen; es gibt ein n, sodass für alle Wörter $x \in L_1$ mit $|x| \geq n$ gilt: „Ich kann x zerlegen in x = uvw mit“:

$|v| \geq 1$

$|uv| \leq n$

$uv^i w \in L_1$ für alle $i \in N_0$

Wir wählen ein bestimmtes Wort mit $m \geq n$ à $x = a^m b^m$ mit m ≥ n à

$|x| = |a^m b^m| = 2m > m \geq n$ à erfüllt

Nach dem Pumping Lemma kann uv so gewählt werden, dass $|uv| \leq n$, also bestehen u und v nur aus dem Buchstaben a.

(1) $u = a^p$ (2) $v = a^q$ mit $q \geq 1; p + q \leq n$ (3) $w = a^r b^m$

$\Rightarrow p + q + r = m$

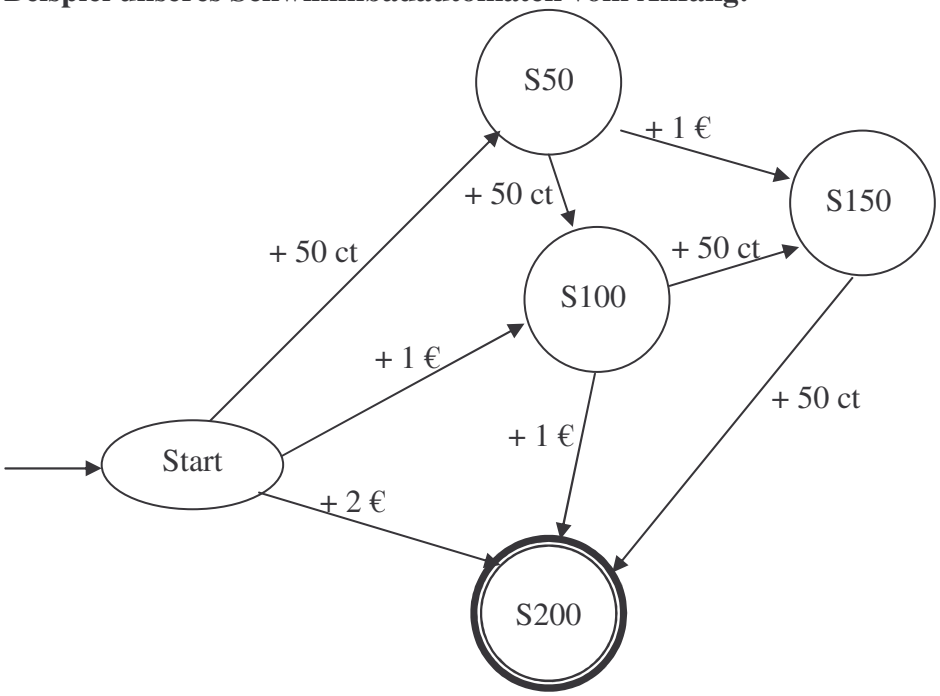
$x = \underbrace{a^p}_{u} + \underbrace{a^q}_{v} + \underbrace{a^r + b^m}_{w}$

Wähle z.B.: i = 0 à $uv^0 w = a^p a^r b^m \notin L_1$ weil $p + r \neq m$

Ergebnis: Die Annahme $L_1 \in \mathrm{Re}\, g_\Sigma$ ist falsch.

Endliche Maschinen

Beispiel unseres Schwimmbadautomaten vom Anfang:



Ausgabe:

Eintrittskarte und Wechselgeld: Karte, Karte + 50, Karte + 100, Karte + 150

Noch zu zahlender Betrag: -50, -100, -150, -200

Ausgabealphabet: $\Delta = \{-50, -100, -150, -200, Karte, Karte+50, Karte+100, Karte+150\}$

Eingabealphabet: $\Sigma \cup \{\text{Karte nehmen}\}$

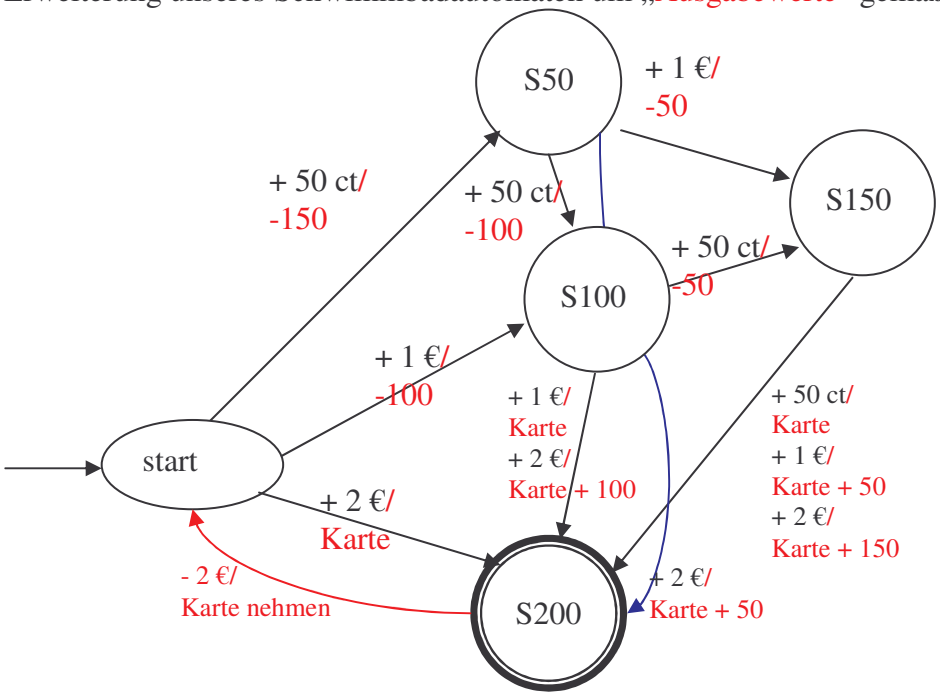
Ausgabefunktion: $\lambda : S \times \Sigma \to \Delta; \lambda(s,a) = b \quad s \in S; a \in \Sigma; b \in \Delta$

λ	50	100	200	Karte nehmen
start	-150	-100	Karte	-
s_{50}	-100	-50	Karte + 50	-
s_{100}	-50	Karte	Karte + 100	-
s_{150}	Karte	Karte + 50	Karte + 150	-
s_{200}	-	-	-	-200 (und zurück zum Start

$\delta(s,a) = s'$

$\lambda(s,a) = b; b \in \Delta$

Erweiterung unseres Schwimmbadautomaten um „Ausgabewerte“ gemäß der Tabelle:

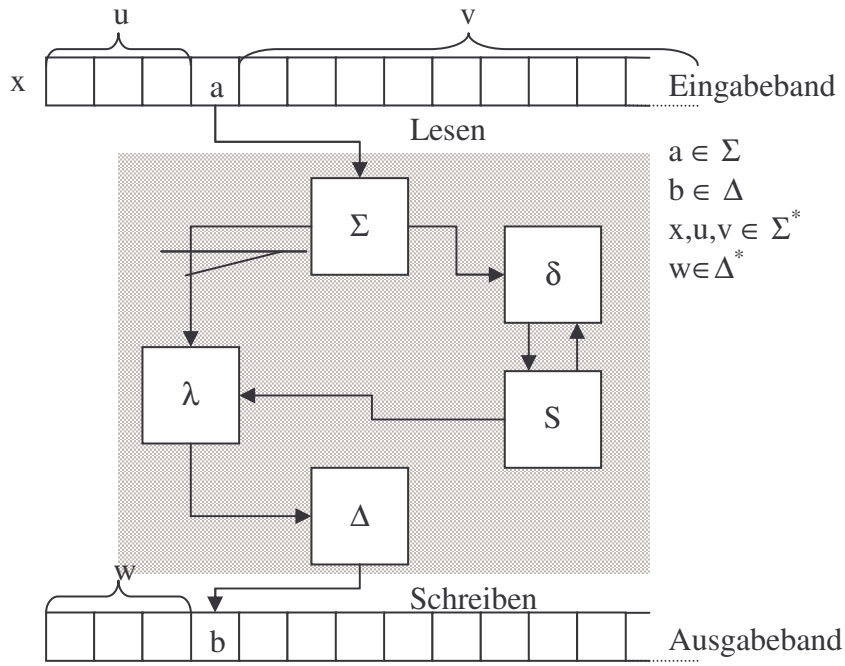


Definition: (Mealy-Maschinen) $M = \{\Sigma, \Delta, S, \delta, \lambda, s_0\}$

Σ	Eingabealphabet
Δ	Ausgabealphabet
S	Endliche Zustandsmenge
δ	Zustandsüberführungsfunktion $\delta = S \times \Sigma \rightarrow S$
λ	Ausgabefunktion $\lambda = S \times \Sigma \rightarrow \Delta$
s_0	Startzustand

Beispiel:

$$M_{swim} = \begin{Bmatrix} \{\underline{50}, \underline{100}, \underline{200}, \underline{\text{Karte nehmen}}\}, \{\underline{-50}, \underline{-100}, \underline{-150}, \underline{-200}, \underline{\text{Karte}}, \underline{\text{Karte}+50}, \underline{\text{Karte}+100}, \underline{\text{Karte}+150}\} \\ \{start, s_{50}, s_{100}, s_{150}, s_{200}\}, \delta, \lambda, start \end{Bmatrix}$$



Eingabewörter werden in Ausgabewörter transformiert.

Konfiguration:

$k = (s, v, w) \in S \times \Sigma^* \times \Delta^*$

s = aktueller Zustand; v = noch zu verarbeitendes Suffix der Eingabe;

w = Bisher erzeugtes Präfix des Ausgabewortes;

Konfigurationsübergang:

$(s, av, w) \mapsto (s', v, wb)$ genau dann, wenn $\delta(s,a) = s'; \lambda(s,a) = b$

$\mapsto^*$ sei wieder reflexive, transitive Hülle von $\mapsto$

Beispiel: M_{swim}

$(s_0, \underline{50}\,\underline{100}\,\underline{100}, \varepsilon) \mapsto (s_{50}, \underline{100}\,\underline{100}, \underline{-150}) \mapsto (s_{150}, \underline{100}, \underline{-150}\,\underline{-50}) \mapsto (s_{200}, \varepsilon, \underline{\text{Karte}+50}\,\underline{-150}\,\underline{-50})$

Berechnung von Mealy-Auromaten:

$f_n = \Sigma^* \to \Delta^*$

Ausgabefunktion λ erweitern:

$\lambda^*(S \times \Sigma^*) \to \Delta^*$

$\lambda^*(s, \varepsilon) \to \varepsilon$

$\lambda^*(s, av) = \lambda(s,a) \circ \lambda^*([\delta(s,a)], v)$

Definition: Zu einer beliebigen Mealy-Maschine $M = \{\Sigma, \Delta, S, \delta, \lambda, s_0\}$

Die Funktion $f_n = \Sigma^* \to \Delta^*$ definiert durch $f_n(x) = \lambda^*(s_0, x)$ heißt „die von M berechnete (Wort)Funktion“.

Beispiel: $M_{swim}; f_{m\,mswim} = (\underline{50}, \underline{100}, \underline{100}) = \lambda^*(s_0, \underline{50}\,\underline{100}\,\underline{100}) = (\underline{-150}, \underline{-50}, \underline{\text{Karte}+50})$

Beispiel: duale Addition

$\Sigma = \{0,1\}$; $plus := \Sigma^+ \times \Sigma^+ \rightarrow \Sigma^+$

plus (x,y) = z genau dann, wenn $wert(x) + wert(y) = wert(z)$

$$wert(x) = wert(x_{n-1}...x_1) = \sum_{i=0}^{n-1} x_i z_i$$

$x_{n-1} \ldots x_0 = 00110$

$y_{m-1} \ldots y_0 = 010110$

1. Auffüllen mit Nullen (kursiv dargestellt)
2. Reihenfolge „spiegeln“, damit LSB (Least significant Bit) zuerst bearbeitet werden kann
3. Wir führen ein neues Eingabealphabet ein

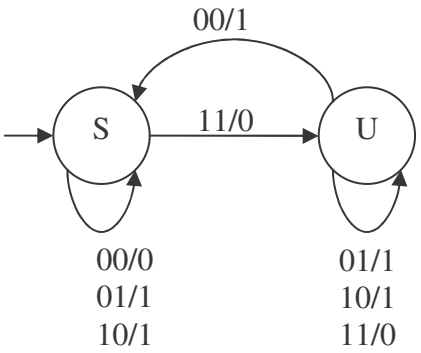
   $\Sigma' = \{\underline{00}, \underline{01}, \underline{10}, \underline{11}\}$, also

   $$\left.\begin{matrix} 001110 \\ 010110 \end{matrix}\right\} \begin{matrix} \underline{00}\,\underline{01}\,\underline{10}\,\underline{11}\,\underline{11}\,\underline{00} \\ \underline{00}\,\underline{11}\,\underline{11}\,\underline{10}\,\underline{01}\,\underline{00} \\ \uparrow\ LSB \end{matrix} \quad \leftarrow \text{gespiegelter (umgedrehter) Wert}$$

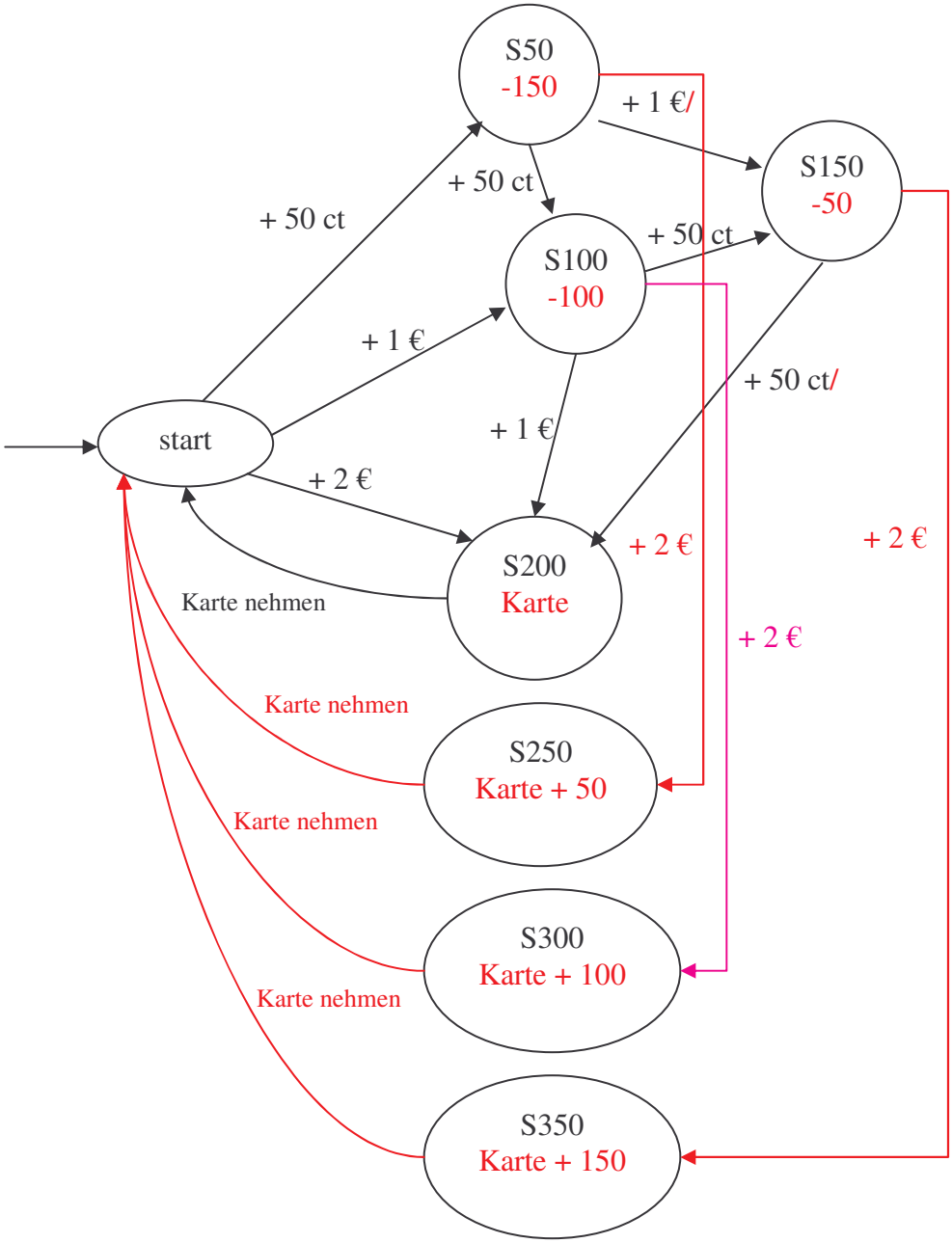
   Startzustand: S (= Zustand „ohne“ Übertrag)

   Weiterer Zustand: U (= Zustand „mit“ Übertrag)

Zustand:	S	S	S	S	U	U	U	U
Eingabe:	00	01	10	11	00	01	10	11
Ausgabe:	0	1	1	0	1	0	0	1
Folgezustand:	S	S	S	U	S	U	U	U



$$M_{add} = (\{\underline{00}, \underline{01}, \underline{10}, \underline{11}\}, \{0,1\}, \{S,U\}, \delta, \lambda, S)$$



Definition Moore-Maschine: $M = \{\Sigma, \Delta, S, \delta, \lambda, s_0\}$

Σ	Eingabealphabet
Δ	Ausgabealphabet
S	Endliche Zustandsmenge
δ	Zustandsüberführungsfunktion $\delta = S \times \Sigma \rightarrow S$
λ	Ausgabefunktion $\lambda = S \rightarrow \Delta$
s_0	Startzustand

Konfigurationsübergang:

$(s_0, av, w) \mapsto (s', v, wb)$, falls $\delta(s,a) = s'; \lambda(s') = b$

Ausgabe ist bei Moore-Maschine ein Zeichen länger als die Eingabe.

Beispiel Moore-Maschine:
Eingabe: Bitfolge
Ausgabe: 1, falls der bis dahin gelesene Präfix eine durch drei teilbare Zahl darstellt, sonst 0;

$\Sigma = \{0,1\}; div3 = \Sigma^* \to \Sigma^*$

$div3(x_n...x_1) = 1y_1...y_n$

$$y_i = \begin{cases} 1 \mid wert(x_n...x_{n-i-1}) \bmod 3 = 0 \\ 0 \mid sonst \end{cases}$$

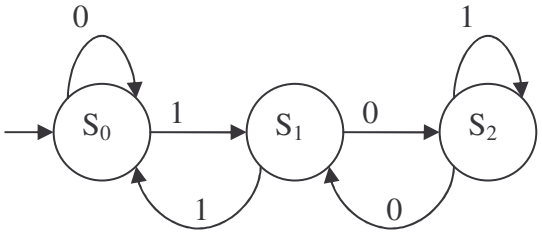
$div3(0111011001101) = 1101100...$

$S_0 \equiv 0 \bmod 3; \quad S_1 \equiv 1 \bmod 3; \quad S_2 \equiv 2 \bmod 3$

Zustand S_0:
Der bisherige Präfix stellt eine durch drei teilbare Zahl dar.
$U \mathrel{\hat{=}} k$
$U0 \mathrel{\hat{=}} 2k$, dann in Zustand S_0 bleiben
$U1 \mathrel{\hat{=}} 2k+1$, dann in Zustand S_1 gehen
$2k+1 \bmod 3 = 1$

Zustand S_1:
$U \mathrel{\hat{=}} k+1, wobei\ k \equiv 0 \bmod 3$
$U0 \mathrel{\hat{=}} 2(k+1) = 2k+2 \equiv 2 \bmod 3 \Rightarrow S_2$
$U1 \mathrel{\hat{=}} 2(k+1)+1 = 2k+3 \equiv 0 \bmod 3 \Rightarrow S_0$

Zustand S_2:
$U \mathrel{\hat{=}} k+2, wobei\ k \equiv 0 \bmod 3$
$U0 \mathrel{\hat{=}} 2(k+2) = 2k+4 \equiv 1 \bmod 3 \Rightarrow S_1$
$U1 \mathrel{\hat{=}} 2(k+2)+1 = 2k+5 \equiv 2 \bmod 3 \Rightarrow S_2$

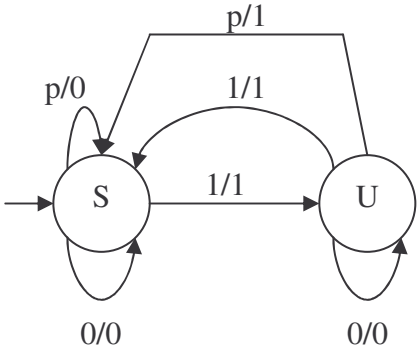


$M = (\{0,1\}, \{0,1\}, \{S_0, S_1, S_2\}, \delta, \lambda, S_0)$

Beispiel Prüfbit:
$Sender \xrightarrow{Nachricht} Empfänger$, binär codiert: $\Sigma = \{0,1\}$; Bits können kippen, z.B.: 101…1101 | 0/1
0, falls Anzahl Einsen gerade ist; 1, falls Anzahl Einsen ungerade ist
à Fehlererkennung (ACHTUNG: Keine Fehlerkorrektur)

$$w \in \{0,1\}; pb(w) = \begin{cases} w1, \text{falls anz } (w,1) = 2k+1 \\ w0, \text{ falls anz } (w,0) = 2k \end{cases}$$

Mealy-Maschine:



S_0 = gerade Anzahl von Einsen
S_1 = ungerade Anzahl von Einsen

Kontextfreie Sprachen

$L_1 = \{a^n b^n \mid n \geq 0\}; \Sigma = \{a, b\}$

$L_1 \notin TYP3_\Sigma$

L_1 erzeugen mit folgenden Regeln :

$$\begin{matrix} S \to aSb \\ S \to \varepsilon \end{matrix} \Rightarrow S \to aSb \to aaSbb \to ... \to a^n b^n$$

$G = (\{a, b\}, \{S\}, \{S \to aSb; S \to \varepsilon\}, S)$

Definition kontextfreie Sprachen:

G = (Σ, N, P, S)
Σ = Eingabealphabet (Terminalzeichen)
N = Nicht-Terminalzeichen
P = endliche Produktionsmenge
S = Startsymbol

$P \subseteq N \times (\Sigma \cup N)^*$

- Eine Sprache L heißt kontextfrei, falls es eine kontextfreie Grammatik G über Σ gibt mit $L_{(G)} = L$
- KfS_Σ bezeichne die Klasse der kontextfreien Sprachen über Σ
- TYP2-Grammatiken sind kontextfreie Grammatiken
  $TYP2_\Sigma = KfS_\Sigma$

Beispiele:

Beschreibung von arithmetischen Ausdrücken. Bezeichnung für Variablen/Konstanten: a
a, a+a, a•a, a•((a+a)/a)
Terminale: a, (,), +, –, •, /
Nicht-Terminale: E (Ausdrücke); O (Operatoren)

$E \to a$

$E \to EOE$

$E \to (E)$

$O \to + \mid - \mid \bullet \mid /$

$G_{arith} = (\{a, (,), +, -, \bullet, /\}, \{E, O\}, P, E)$

$$\begin{aligned} E &\to EOE \\ &\to E + E \\ &\to E + (E) \\ &\to E + (EOE) \\ &\to E + (E / E) \\ &\to E + (E / (EOE)) \\ &\to E + (a / (a + a)) \end{aligned}$$

Beispiel: $G = (\{a, b\}, \{S, A\}, P, S)$

$$P = \begin{matrix} \{S \to aAS \mid a \\ A \to SbA \mid SS \mid ba\} \end{matrix}$$

Linksableitungen:

In jedem Schritt wird das am weitestem links stehende Nicht-Terminal ersetzt.

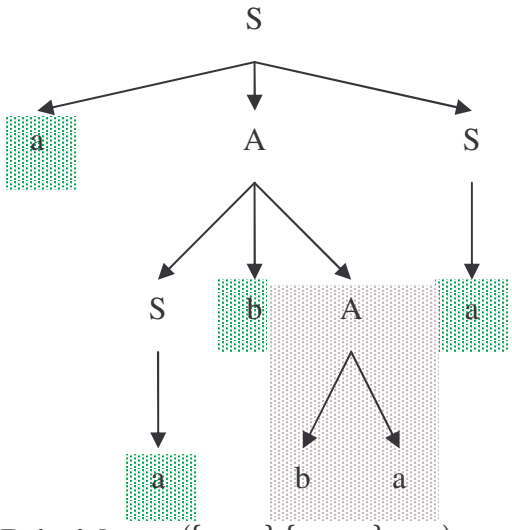
$$\begin{aligned} S &\Rightarrow aAS \\ &\Rightarrow aSbAS \\ &\Rightarrow aabAS \\ &\Rightarrow aabbaS \\ &\Rightarrow aabbaa \end{aligned}$$

Rechtsableitungen:

Mit jedem Schritt wird das am weitestem rechts stehende Nicht-Terminal ersetzt.

$$\begin{aligned} S &\Rightarrow aAS \\ &\Rightarrow aAa \\ &\Rightarrow aSbAa \\ &\Rightarrow aSbbaa \\ &\Rightarrow aabbaa \end{aligned}$$

Ableitungsbaum / Parse Tree



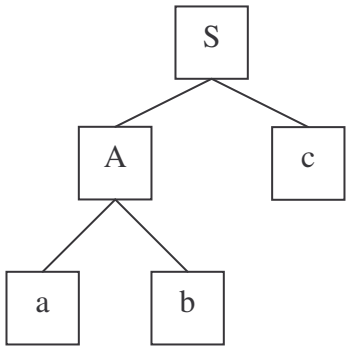
1. Wurzel
2. Innere Knoten (Nicht-Terminale)
3. Jedes Blatt (grün unterlegt) ist mit einem Symbol aus $N \cup \Sigma \cup \{\varepsilon\}$ markiert
4. Für alle Teilbäume der Art des grau unterlegten Bereichs links muss A à ba ∈ P sein

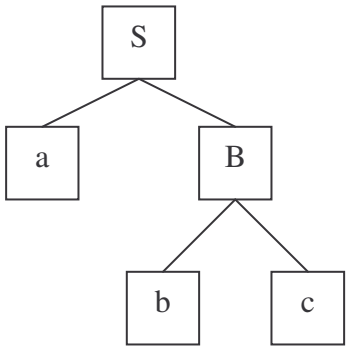
Beispiel: $G = (\{a,b,c\},\{S,A,B\},P,S)$

$P = \{S \to aB \mid Ac$
$A \to ab$
$B \to bc\}$

S à Ac
à abc

S à aB
à abc

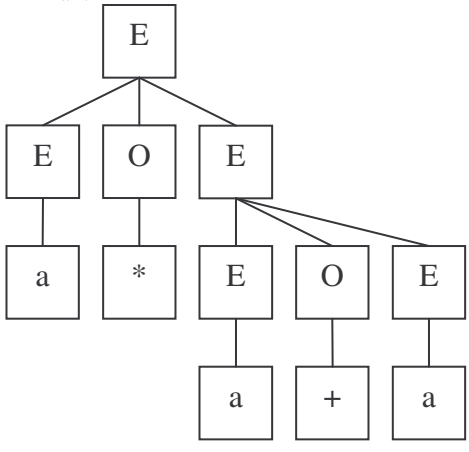


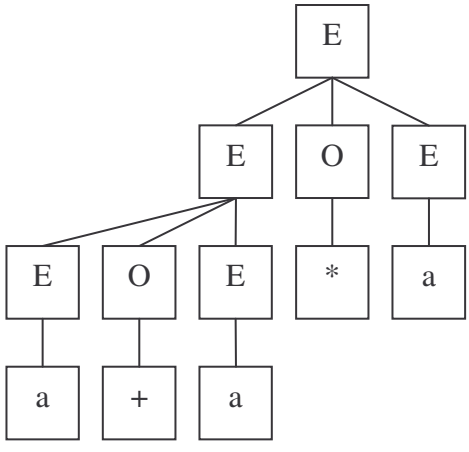


Das Wort abc kann in G auf „wesentlich“ verschiedene Weise Abgeleitet werden, da auch die Ableitungssysteme verschieden sind.

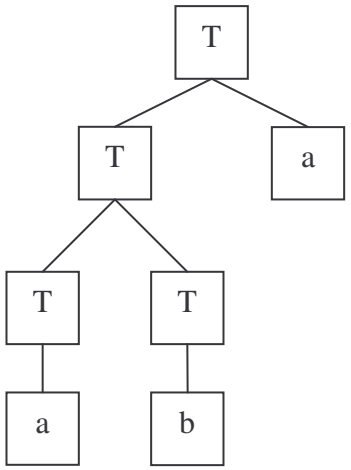
Definition: Eine kontextfreie Grammatik G heißt mehrdeutig, falls es für mindestens ein Wort w ∈ $L_{(G)}$ mindestens zwei verschiedene Ableitungsbäume gibt.

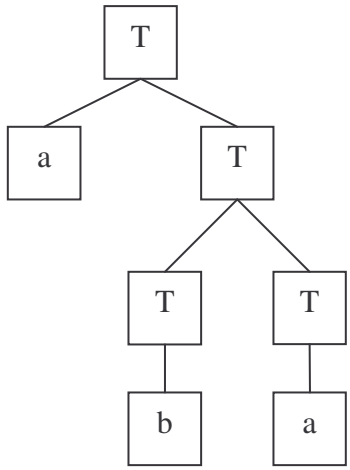
Beispiel: G_{arith}





Beispiel: $G = (\{a,b\},\{T\},\{T \to TT \mid a \mid b\},T)$ = mehrdeutige Grammatik





aba ∈ $L_{(G)} = \{a,b\}^+$

$G' = (\{a,b\},\{T\},\{T \to aT \mid bT \mid a \mid b\},T)$; G' ist nicht mehrdeutig und G ≡ G', d.h. $L_{(G)} = L_{(G')}$

Definition: Eine kontextfreie Grammatik L heißt <u>inhärent mehrdeutig</u>, falls alle Grammatiken, die L erzeugen, mehrdeutig sind. Andernfalls heißt L auch eindeutig.

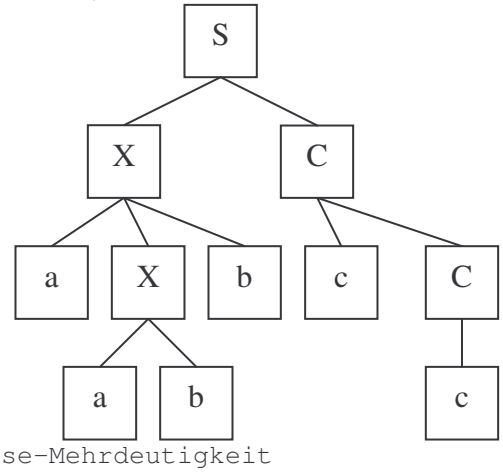
Beispiel: Für eine mehrdeutige Sprache $L = \{a^i b^j c^k \mid i,j,k \geq 1, i = j \text{ oder } j = k\}$

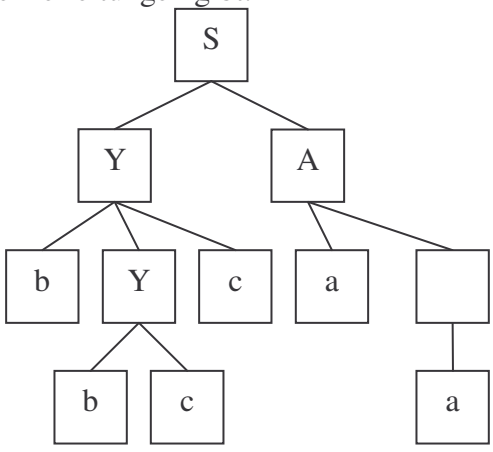
$G = (\{a,b,c\},\{S,A,C,X,Y\},P,S)$

$P = \{S \to XC \mid AY,$
$X \to aXb \mid ab$
$C \to cC \mid c$
$A \to aA \mid a$
$Y \to bYc \mid bc\}$

$aabbcc \in L$

L ist inhärent, da es für Wörter wie aabbcc immer mehrere Herleitungen gibt.



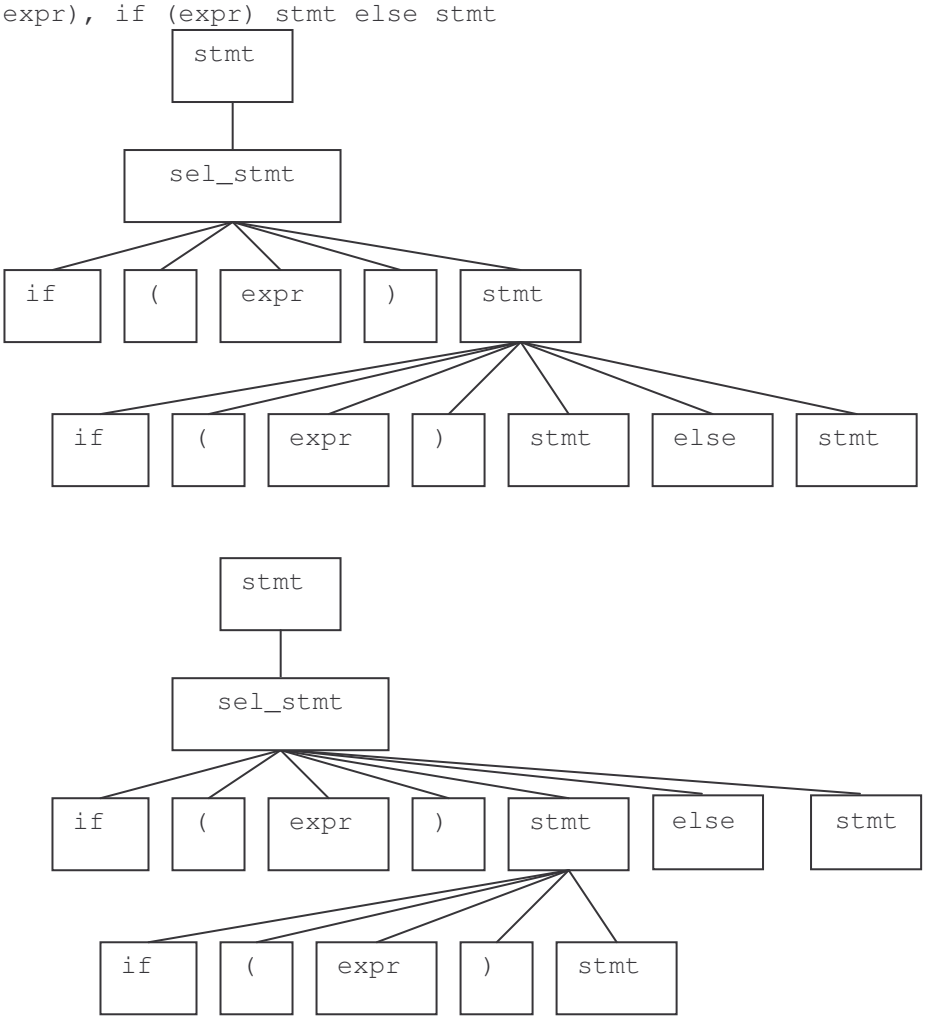



```
if-else-Mehrdeutigkeit
stmt à  sel_stmt
sel_stmt à  if (expr) stmt
sel_stmt à  if (expr) stmt else stmt
```

Terminale: `(, ), if, else`

Nicht-Terminale: `stmt, sel_stmt, expr`

Beispiel: `if (expr), if (expr) stmt else stmt`



Vereinfachung und Normalformen von kontextfreien Grammatiken

1. Eliminierung von ε-Regeln
   Eine kontextfreie Grammatik G heißt ε-frei, falls P keine Regel der Art A à ε, A ∈ N enthält
   Jede kontextfreie Grammatik G kann in eine ε-freie Grammatik G' umgewandelt werden mit $L_{(G)}$ à $L_{(G')}$ bis auf das leere Wort ε, das möglicherweise in $L_{(G)}$ enthalten ist.
2. Algorithmus:
   a. $N_1 = \{A \in N \mid (A \to \varepsilon) = P\}$
      Menge der Nicht-Terminale, die linke Seite einer ε-Regel sind.
   b. Bilde N_2, die Menge aller Nicht-Terminale, aus denen das leere Wort ableitbar ist.
      repeat;
      $N_2 = N_1$
      $N_1 = N_1 \cup \{A \in N \mid A \to w, w \in N_1^*\}$
      until $N_1 = N_2$
3. Für jede Regel, deren rechte Seite ein Nicht-Terminal aus N_2 enthält, fügen wir die gleiche Regel ohne das Nicht-Terminal hinzu. **Beispiel:**

$A \to w_1 B w_2; B \in N_2$

$A \to w_1 w_2$ neu hinzufügen

```
P1 = P
repeat
for each A à w1Bw2 ∈ P2 do
    if B ∈ N2 then
        P1 = P1 ∪ {A à w1w2}
    end if
end for
until P1 = P
```

4. Wir eliminieren alle ε-Regeln, d.h.:
   $P_2 = P_1 - \{r \in P_1 \mid r = A \to \varepsilon\}$
   $G' = \{\Sigma, N, P_2, S\}$
   Falls $\varepsilon \notin L_{(G)}$, dann sind G und G' äquivalent, andernfalls $L_{(G)} = L_{(G')} \cup \{\varepsilon\}$

Eliminierung von Kettenregeln

Kettenregeln sind Regeln der Art A à B, A,B ∈ N.
Kettenregeln können eliminiert werden.
Sei G = {Σ, N, P, S} eine ε-freie kontextfreie Grammatik.

Algorithmus:

1. Zyklen eliminieren:
   $B_1, B_2, ..., B_k \in N$
   $B_i \to B_{i+1} \quad 1 \le i \le k$
   $B_k \to B_1$
   - Dann streiche alle B aus N
   - Nimm ein neues B zu N hinzu
   - Ersetze in allen Regeln von P jedes B_i durch B
2. $N = \{A_1, A_2, ..., A_s\}$ alle Nicht-Terminale so nummerieren, dass für A_i à A_j folgt, wobei i < j
3. Für jedes A_i à A_j mit i < j fügen wir für jedes $A_j = A_i$ à w hinzu, wobei w ∈ $\{N \cup \Sigma\}^*$

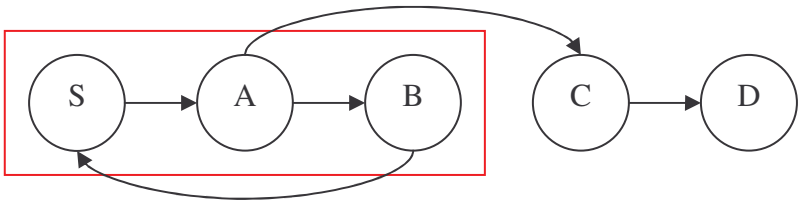
```
for i = s - 1 to 1 do
    for all j > i do
        eliminate Ai à Aj
        if exists Aj à w then
            add Ai à w
        end if
    end for all
end for
```

$w \in (\Sigma \cup N)^* \setminus N$

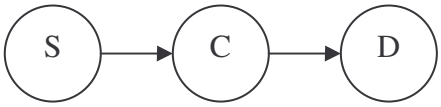
Beispiel:

$G = (\{a,b,c,d\}, \{S, A, B, C, D\}, P, S)$

$P = \{S \to A \mid aB \mid aC$
$B \to S \mid Ba$
$D \to d \mid dDD$
$A \to B \mid C \mid cAd$
$C \to D \mid c\}$



$P' = \{S \to aS \mid aC \mid Sa \mid C \mid cSd$
$D \to d \mid dDD$
$C \to D \mid c\}$



$P'' = \{S \to aS \mid aC \mid Sa \mid d \mid dDD \mid c \mid cSd$
$D \to d \mid dDD$
$C \to d \mid dDD \mid c\}$

$G'' = (\{a,b,c,d\}, \{S, C, D\}, P, S)$ ist äquivalent zu G und ohne Kettenregeln.

Chomsky-Normalform

Definition: Sei $G=(\Sigma,N,P,S)$ eine kontextfreie Grammatik, dann ist G in Chomsky-Normalform (CNF), falls $P\subseteq N\times((N^{\circ}N)\cup\Sigma)$. Beispiel:

$A\to BC \quad A,B,C\in N$

$A\to a \quad a\in\Sigma$

Es gibt zu jeder kontextfreien Grammatik $G=(\Sigma,N,P,S)$ mit $\varepsilon\notin L(G)$ eine äquivalente Grammatik G' in CNF.

Algorithmus:

Eingabe: ε-freie Grammatik G ohne Kettenregeln

Ausgabe: äquivalente Grammatik G' in Chomsky-Normalform

Alle Regeln, die noch nicht in Chomsky-Normalform sind, haben folgende Gestalt:

* A à $X_1X_2\ldots X_m$; $m>2$; $X_i\in N\cup\Sigma$

1. **Schritt:** $X_i\in\Sigma$, dann ersetzen wir X_i in * durch ein neues Nicht-Terminal (X_i und ergänzen P um C_{Xi} à X_i und N um C_{xi} erweitern
2. **Schritt:** Jetzt haben alle Regeln, die nicht in Chomsky-Normalform sind, folgende Gestalt:
   A à $B_1B_2\ldots B_m$; $B_i\in N$
   Jede Regel dieser Bauart ersetzen wir durch:
   A à B_1D_1 Mit D neu in N
   D_1 à B_2D_2
   D_{m-3} à $B_{m-2}D_{m-2}$
   D_{m-2} à $B_{m-1}B_m$

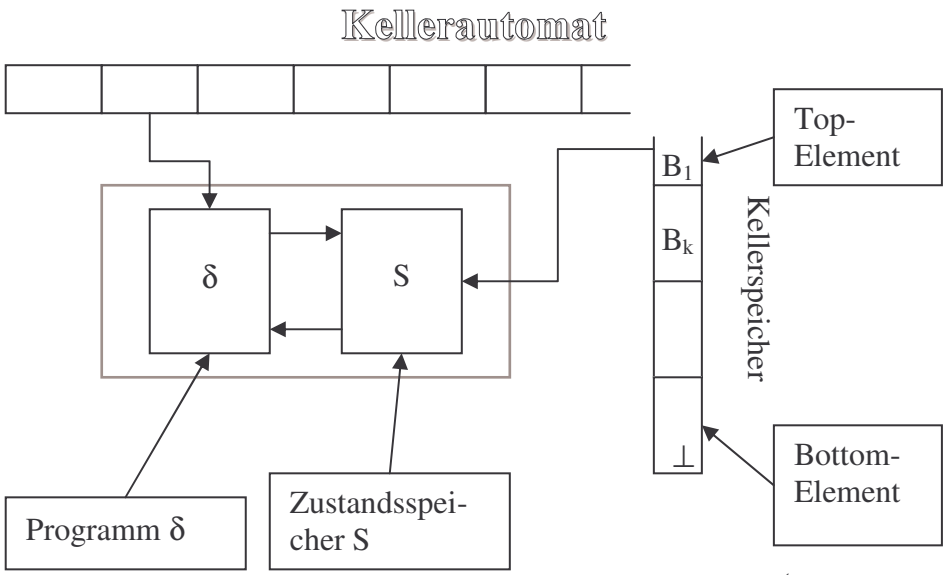
   Beispiel: $G=(\{a,b\},\{S,A,B\},P,S)$

$P=\{S\to bA,\ S\to aB,\ A\to bAA,\ A\to aS,\ A\to a,\ B\to aBB,\ B\to bS,\ B\to b\}$

$P'=\{S\to BA,\ S\to AB,\ A\to BAA,\ A\to AS,\ A\to a,\ B\to ABB,\ B\to BS,\ B\to b\}$

$P''=\{S\to BA,\ S\to AB,\ A\to BC,\ C\to AA,\ A\to AS,\ A\to a,\ B\to AD,\ D\to BB,\ B\to BS,\ B\to b\}$

$G''=(\{a,b\},\{S,A,B,C,D\},P'',S)$ ist in Chomsky-Normalform und $G''\cong G$



Definition: Ein (nicht-deterministischer) Kellerautomat ist ein System $K = (\Sigma, S, \Gamma, \delta, s_0, \perp, F)$

Σ = Eingabealphabet
S = Endliche Zustandsmenge
Γ = Kelleralphabet
δ = Zustandsüberführungsfunktion à $\delta = S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to P(S \times \Gamma^*)$
s_0 = Startzustand
$\perp$ = Kellet-Button-Symbil ($\perp \in \Gamma$)
$F \subseteq S$ = Menge der Endzustände
$(s', B_1 ... B_k) \in \delta\ (s, a, A)$

$s \in S, a \in \Sigma, A \in \Gamma, s \in S, B_1 ... B_k \in \Gamma^*$

Wenn sich K im Zustand s befindet, das Zeichen a einließt und A das Top-Element im Kellerspeicher ist, dann <u>kann</u> K in den Zustand s übergehen und dabei A durch das Wort $B_1 \ldots B_k$ ersetzen, wobei B_1 das neue Top-Element ist.

$\delta(s, a, A) = \{(s_1 \gamma_1) ... (s_n \gamma_n)\};\ s_i \in S, \gamma_i \in \Gamma^*$

Wir schreiben dies auch als Programmzeile: $(s, a, A), \{(s_1 \gamma_1) ... (s_n \gamma_n)\}$

a = ε: für a = ε wird kein Eingabesymbol gelesen, sondern nur eine Manipulation des Kellerspeichers vorgenommen. Falls für $\gamma = B_1 ... B_k; B_i \in \Gamma$ mit k = 0, wird nur das Top-Element gelöscht.

Beispiel: $L_1 = (a^n b^n \mid n \geq 0)$

$K_1 = (\{a, b\}, \{s_0, s_1, s_F\}, \{\perp, a\}, \delta, s_0, \perp, \{s_F\})$

$\delta_1 = \{(s_0, \varepsilon, \perp, s_F, \varepsilon)$
$(s_0, a, \perp, s_0, a \perp)$
(s_0, a, a, s_0, aa)
$(s_0, b, a, s_1, \varepsilon)$
$(s_1, b, a, s_1, \varepsilon)$
$(s_1, \varepsilon \perp, s_F, \varepsilon)$

Konfiguration:

k = (s,w,α)

s = aktueller Zustand | w = noch zu verarbeitendes Suffix der Eingabe | α = sktueller Kellerinhalt

$(s, av, A\alpha) \mapsto (s', v, \beta\alpha)$ ist gültiger Konfigurationsübergang, wenn $(s', \beta) \in \delta(s, a, A)$ ist.

s,s' ∈ S | v ∈ Σ^* | α,β ∈ Γ^*

a ∈ Σ | A ∈ Γ

Sei $K = (\Sigma, S, \Gamma, \delta, s_0, \perp, F)$ ein Kellerautomat, dann ist $L_F(K) = \{w \in \Sigma^* \mid (s_0, w, \perp) \mapsto^* (s_F, \varepsilon, \gamma)\}$ mit $s_F \in F, \gamma \in \Gamma^*$ die von K mit Endzustand über Σ akzeptierte Sprache.

Bez.:

PDA^F_Σ = Klasse der von Kellerautomaten mit Endzustand akzeptierten Sprachen

Beispiel:

$L_2 = \{wcSP(w) \mid w \in \{a,b\}^*, SP = Spiegelung\}$

$\Sigma = \{a,b,c\}$ K_2 konstruieren

$K_2 = (\{a,b,c\}, \{s_0, s_c, s_F\}, \{a,b,\perp\}, \delta_2, s_0, \perp, \{s_F\})$

$$\delta_2 = \left\{ \begin{array}{l} (s_0,a,\perp,s_0,a\perp),(s_0,b,\perp,s_0,b\perp), \\ (s_0,a,a,s_0,aa),(s_0,b,b,s_0,bb), \\ (s_0,a,b,s_0,ab),(s_0,b,a,s_0,ba), \\ (s_0,c,a,s_c,a),(s_0,c,b,s_0,b), \\ (s_0,c,\perp,s_F,\varepsilon), \\ (s_c,a,a,s_c,\varepsilon),(s_c,b,b,s_c,\varepsilon), \\ (s_c,\varepsilon,\perp,s_c,\varepsilon) \end{array} \right\}$$

$L_3 = \{wSP(w) \mid w \in \{a,b\}^*, SP = Spiegelung\}$

$\Sigma = \{a,b\}$

$G_3 = (\{a,b\}, \{S\}, \{S \to aSa \mid bSb \mid \varepsilon\}, S)$

$K_3 = (\{a,b\}, \{s_0, s_c, s_F\}, \{a,b,\perp\}, \delta_3, s_0, \perp, \{s_F\})$

$$\delta_2 = \left\{ \begin{array}{l} (s_0,c,\perp,s_F,\varepsilon),(s_0,a,\perp,s_0,a\perp), \\ (s_0,b,\perp,s_0,b\perp), \\ (s_0,a,a,\{(s_c,\varepsilon),(s_0,aa)\}), \\ (s_0,b,b,\{(s_c,\varepsilon)(s_0,bb)\}), \\ (s_0,a,b,s_0,ab),(s_0,b,a,s_0,ba), \\ (s_c,a,a,s_c,\varepsilon),(s_c,b,b,s_c,\varepsilon), \\ (s_c,\varepsilon,\perp,s_c,\varepsilon) \end{array} \right\}$$

Beispiel für eine Eingabe:

abbbba ∈ L_3

$(s_0, abbbba, \perp) \mapsto (s_0, a, bbbba, a\perp) \mapsto (s_0, b, bbba, ba) \mapsto$ Auswahl 1 $(s_c, b, bba, a\perp)$ Konfigurationsfolge stoppt hier, da kein Endzustand erreicht wird und die Eingabe nicht vollständig abgearbeitet ist

à Back-Tracking, also folgt:

$(s_0, b, bbba, ba) \mapsto (s_0, b, bba, bba\perp) \mapsto$ Auswahl 2 $(s_c, b, ba, ba\perp) \mapsto (s_c, b, a, a\perp) \mapsto (s_c, a, \varepsilon, \perp) \mapsto (s_F, \varepsilon, \varepsilon, \varepsilon)$

Definition:

$K = (\Sigma, S, \Gamma, \delta, s_0, \perp, F)$ sei ein Kellerautomat, dann heißt $L_\varepsilon(K) = \{w \in \Sigma^* \mid (s_0, w, \perp) \mapsto^* (s, \varepsilon, \varepsilon)\}$

s = Zustand beliebig die von K mit leerem Keller akzeptierte Sprache

PDA^ε_Σ = Klasse der mit leerem Keller akzeptierten Sprachen. F ist nicht relevant.

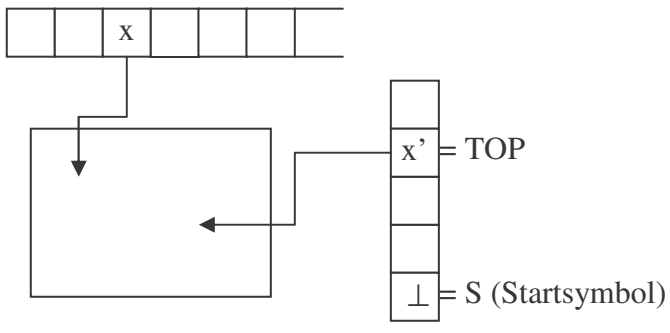
Bez.: $PDA_\Sigma = PDA^F_\Sigma = PDA^\varepsilon_\Sigma$

Klasse der mit Kellerautomaten beschreibbaren Sprachen $KfS_\Sigma = PDA_\Sigma$

$KfS_\Sigma \supseteq PDA_\Sigma$ à ohne Beweis für die Praxis

$KfS_\Sigma \subseteq PDA_\Sigma$ à Zu jeder kontextfreien Grammatik $G = (\Sigma, N, P, S)$ lässt sich ein Kellerautomat K konstruieren, daher $L_{(G)} = L_{(K)}$

Wir konstruieren K derart, dass K die Linksableitung von G simuliert.



- $x' \in \Sigma$ und $x = x'$:
  x' löschen
  Lesekopf auf Eingabeband ein Feld weiter rücken
- $x' = A \in N$
  kein Eingabezeichen lesen
  ε-Übergang
  A ersetzen durch α mit $A \to \alpha \in P$
- $\perp$ = S (Startsymbol aus G)
- Kellerautomat enthält Terminale und Nicht-Terminale aus G
- Kellerautomat verwendet hier nur einen Zustand

Formal

$K = (\Sigma, \{s\}, \Sigma \cup N, \delta, s, \perp = S)$

$\delta = \{(s, a, a, s, \varepsilon) \mid a \in \Sigma\} \cup \{(s, \varepsilon, A, s, \alpha) \mid A \to \alpha \in P\}$

Es gilt $L_{\varepsilon(K)} = L_{(G)}$ = Akzeptieren mit leerem Keller. Dieser Kellerautomat wird daher auch „PARSER“ genannt.

$L_1 = \{a^n b^n \mid n \geq 0\}$

$G_1 = (\{a,b\}, \{s\}, \{S \to aSb \mid \varepsilon\}, s)$

Parser für G:

$K_G = (\{a,b\}, \{s\}, \{a,b,S\}, \delta, \perp = S)$

$\delta = \{(s_0, a, a, s_0, \varepsilon), (s_0, b, b, s, \varepsilon), (s_0, \varepsilon, S, \{(s_0, aSb) \mid (s_0, \varepsilon)\})\}$

Beispiel für eine Eingabe: $a^3b^3 \in L_1$

$(s_0, a^3b^3, S) \mapsto$

(s_0, a^3b^3, aSb) ← *Auswahl*

$\mapsto (s_0, a^2b^3, Sb)$

$\mapsto (s_0, a^2b^3, aSbb)$ ← *Auswahl*

$\mapsto (s_0, a^1b^3, Sbb)$

$\mapsto (s_0, a^1b^3, aSbbb)$ ← *Auswahl*

$\# \mapsto (s_0, b^3, Sbbb)$

$\mapsto (s_0, b^3, aSbbb)$ ← *Auswahl*

$\mapsto (ERROR)$ → Back – Tracking nach #

$\# \mapsto (s_0, b^3, Sbbb)$

$\mapsto (s_0, b^3, bbb)$ ← *Auswahl*

$\mapsto (s_0, b^2, bb)$

$\mapsto (s_0, b^1, b)$

$\mapsto (s_0, \varepsilon, \varepsilon)$

$(s_0, a^3b^3, S) \mapsto^* (s_0, \varepsilon, \varepsilon)$

Deterministische Kellerautomaten

Parser: Bei kontextfreien Grammatiken ein nicht-deterministischer Kellerautomat.

→ Back-Tracking kann erforderlich werden

→ Exponentielle Laufzeit, da ggf. der gesamte Berechnungsbaum durchsucht werden muss

Definition:

Ein Kellerautomat $K = (\Sigma, S, \Gamma, \delta, s_0, \perp, F)$ heißt deterministisch, falls $|\delta(S, a, A)| + |\delta(s, \varepsilon, A)| \leq 1$ für alle $s \in S; a \in \Sigma; A \in \Gamma$

Eine Sprache L heißt deterministisch und kontextfrei, wenn es einen deterministischen Kellerautomat K gibt, der L akzeptiert, d.h. für $L_{(K)} = L$

$L_{(K)} = \{w \in \Sigma \mid (s_0, w, c) \mapsto^* (s_F, \varepsilon, \gamma), s_F \in F, \gamma \in \Gamma^*\}$

Bezeichnung: $DPDA_\Sigma$

$L_3 = \{(wSPcw) \mid w \in \{a,b\}^*\}$

aaabbbaaabbb…

$L_3 \notin DPDA_\Sigma$!

$DPDA_\Sigma \neq PDA_\Sigma$

$DPDA_\Sigma$: hier kann sehr effizient in „linearer Zeit“ für jede Eingabe die Zugehörigkeit zu der beschriebenen Sprache festgestellt werden. Für viele praktische Anwendungen reicht $DPDA_\Sigma$ vollkommen aus.

Pumping-Lemma für KfS:

Sei L eine Kontextfreie Sprache, dann gibt es eine Zahl n, sodass alle Wörter $z \in L$ mit $|z| \geq n$ gilt.

z = uvwxy mit folgenden Eigenschaften:

1. $|vw| \geq 1$
2. $|vwx| \leq n$
3. $uv^i wx^i y \in L$

Beweis:

Sei G eine Kontextfreie Grammatik in CNF (A à BC | a), die L erzeugt

$m = |n|$

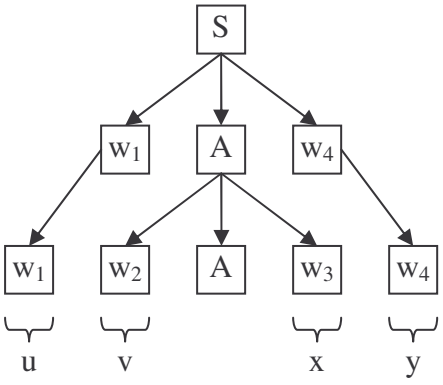
Wähle $n = 2^m$

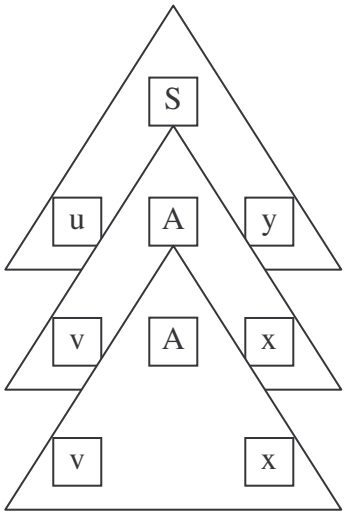
$z \in L; |z| \geq n$

Ableitungsbaum hat sicher $\geq 2^m$ viele Knoten. Also a gibt ein Nicht-Terminal, das auf einem Pfad zwei Mal auftritt.

$S \to w_1 A w_4$

$\to w_1 w_2 A w_3 w_4$





$L = \left(a^k b^k a^k \mid k > 0\right)$
$z = a^k b^k a^k$

L $\notin$ KfS_Σ, da die Regel 3 des Pumping Lemma verletzt wird.

Syntax von Programmiersprachen

Erweiterte Baccus-Naur-Form:

1. Sei bereits abgeleitet eine EBNF folgender Form:
   $w_1(\alpha_1 \mid ... \mid \alpha_k)w_2$, dann gilt $\Rightarrow w_1 a_i w_2$ für $1 \le i \le k$
2. Es sei bereits abgeleitet:
   $w_1\{\beta\}^* w_2$, dann gilt $w_1 \beta^i w_2$ für $i \in N_0$
3. Sei bereits abgeleitet:
   $w_1[\beta]w_2$ dann gilt entweder $w_1 \beta w_2$ oder $w_1 w_2$ (alternativ : $w_1\{\beta\}_1^0 w_2$ oder $w_1|\beta|w_2$)
   Dies ist zwar auch durch Punkt 2 abgedeckt, jedoch findet speziell diese Definition beispielsweise bei den If-Schleifen Verwendung. Hier eine beispielhafte Darstellung, um das zu verdeutlichen:
   `If (...) then (...) [else (...)]`
   Eine If-Schleife kann einen Else-Teil haben, muss sie aber nicht. Auch ist es bei If-Schleifen nicht möglich, zwei Else-Teile anzulegen!

Erweiterte Baccus-Naur-Grammatik:

$L_{(G)} = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$

Beispiel:

$G_{Integer} = (\{+,-,0,1,2,3,4,5,6,7,8,9\}, \{<\text{int}>, <vz>, <zf>, <z1>, <z0>\}, P, <\text{int}>)$

$$P = \left\{ \begin{array}{l} <\text{int}> ::= <vz><zf> \\ <vz> ::= [(+,-)] \\ <zf> ::= (0 \mid <z1>\{<z0>\}^*) \\ <z1> ::= (1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9) \\ <z0> ::= (0 \mid <z1>) \end{array} \right\}$$

Ableitungsbeispiel nach dieser Grammatik für die Integer-Zahl +125:

$$\begin{aligned} <\text{int} &\Rightarrow <vz><zf> \\ &\Rightarrow [(+\mid -)] <zf> \\ &\Rightarrow (+\mid -) <zf> \\ &\Rightarrow + <zf> \\ &\Rightarrow +(0 \mid <z1>\{<z0>\}^*) \\ &\Rightarrow + <z1><z0><z0> \\ &\Rightarrow +(1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9) <z0><z0> \\ &\Rightarrow^* +(1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)(0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)(0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9) \\ &\Rightarrow^* +125 \end{aligned}$$

Bemerkung:

Die von ISO standardisierte Beschreibungssprache ASN1 baut auf EBN-Grammatiken auf.

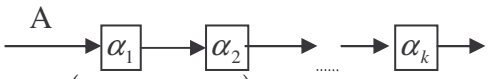
Satz:

$KfS = EBNF_\Sigma$ ß Klasse aller Sprachen, die mit EBN-Grammatien beschrieben werden.

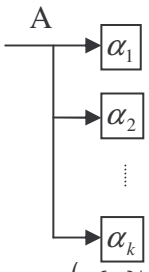
Syntaxdiagramme von Programmiersprachen

Hier am Beispiel von EBN-Grammatien

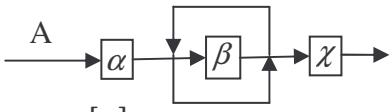
a) $A ::= \alpha_1, \alpha_2, ..., \alpha_k$



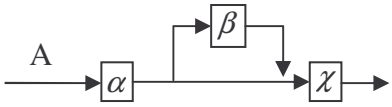
b) $A ::= (\alpha_1 \mid \alpha_2 \mid ... \mid \alpha_k)$



c) $A ::= (\alpha\{\beta\}^* \chi)$



d) $A ::= \alpha[\beta]\chi$



Bemerkung:

Sind α, β, χ oder α^i Terminalsymbole, dann werden sie mit Kreisen oder Ovalen dargestellt. Beispiel hierfür anhand $G_{integer}$: